[Return to the USPTO NPL Page](#) | [Help](#)

Interface language:

[English](#)

Databases selected: Multiple databases...

[What's new](#)

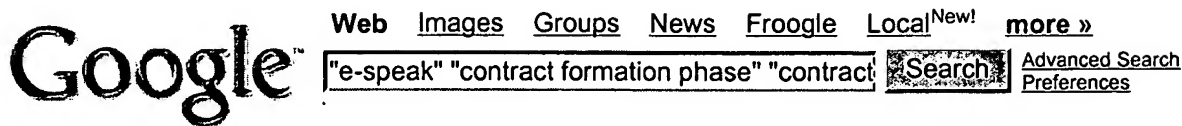
Searching for *PDN(<02/13/2002)* and *"contract formation phase"* and *"contract execution phase"* and *HP* did not find any documents. Try the following:

Revise your search below using the following tips:

- Check your spelling.
- Reduce the number of terms included in your search.
- Broaden your search by selecting other [databases](#), removing limits, or searching "Citations and Document Text" (if available).
- Use "AND" to connect two words that don't need to be searched as a phrase.
- Connect similar terms with the "OR" operator (e.g. military OR pentagon). See [Search Tips](#) for more hints.

Basic Search

Tools: [Search Tips](#) [Browse Topics](#) [1 Recent Searches](#)Database: [Select multiple databases](#)Date range: Limit results to: ☒ Full text documents only ☐ Scholarly journals, including peer-reviewed [About](#)[More Search Options](#)Copyright © 2005 ProQuest Information and Learning Company. All rights reserved. [Terms and Conditions](#)[Text-only interface](#)



Web Results 1 - 2 of 2 for **"e-speak" "contract formation phase" "contract execution phase"**. (0.70 second)

Tip: Try removing quotes from your search to get more results.

[PDF] [Position Papers for the World Wide Web Consortium \(W3C\) Workshop ...](#)

File Format: PDF/Adobe Acrobat - [View as HTML](#)

contract execution phase. The template typically has a number of free variables that

... **Contract formation phase** - Participants assume contract roles and ...

www.hpl.hp.com/techreports/2001/HPL-2001-73.pdf - [Similar pages](#)

[PS] 1 - Introduction The W3C Web Services Workshop represents a ...

File Format: Adobe PostScript - [View as Text](#)

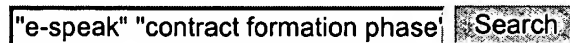
E-speak was designed for doing business in the dynamic environment of the Internet.

... **Contract formation phase** - Participants assume contract roles and ...

www.hpl.hp.com/techreports/2001/HPL-2001-73.ps - [Similar pages](#)



Free! Instantly find your email, files, media and web history. [Download now.](#)



[Search within results](#) | [Language Tools](#) | [Search Tips](#) | [Dissatisfied? Help us improve](#)

[Google Home](#) - [Advertising Programs](#) - [Business Solutions](#) - [About Google](#)

©2005 Google

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S2	7	(videosdotcom or (vesta adj2 broadband))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 14:40
S3	3	(Javed near2 Shoeb).in. or (Tuder near2 John).in. or (Adivi near2 Venkatesh).in.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 14:01
S4	2	("6496932").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 15:02
S6	111	video adj2 download	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 15:03
S7	44	(video adj2 download) and (rental rent buy sell sale purchase)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 16:03
S8	13	(video adj2 download) and ((rental rent buy sell sale purchase) near5 (video or film or movie))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 16:47
S9	221	("705"/\$\$.ccls. and (@ad<"19990913").ad. and ((distribute rental rent buy sell sale purchase) near5 (audio music video or film or movie))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 17:37
S10	177	((("705"/\$\$.ccls. and (@ad<"19990913").ad. and ((distribute rental rent buy sell sale purchase) near5 (audio music video or film or movie)))) and (network internet client server)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 16:50
S11	23	((("705"/\$\$.ccls. and (@ad<"19990913").ad. and ((distribute rental rent buy sell sale purchase) near5 (audio music video or film or movie)))) and (network internet client server)) and (download near10 (video or content))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 16:51

S12	26	((("705"/\$\$.ccls. and (@ad<"19990913").ad. and ((distribute rental rent buy sell sale purchase) near5 (audio music video or film or movie))) and (compression and extract)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 17:41
S13	2	("5,841,979").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 17:45
S14	2	("6,134,558").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 17:46
S15	2	("6,134,548").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/09 17:46
S16	535	(@ad<"20000913").ad. and download and (video near10 (sales distribution distribute rent rental))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 10:57
S17	263	((@ad<"20000913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 10:58
S18	72	(((@ad<"20000913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 11:23
S19	161	"5172413".URPN.	USPAT	OR	OFF	2003/05/12 11:03
S20	26	(((((@ad<"20000913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:19

S21	26	(((((@ad<"20000913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:31
S22	22	(((((@ad<"19990913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:34
S23	22	(((((@ad<"19990913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL)) and (subscriber or subscriber)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:33
S24	22	(((((@ad<"19990913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL)) and (download transmit)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:33
S25	22	(((((@ad<"19990913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL)) and (payment amount pay)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:33
S26	16	(((((@ad<"19990913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL)) and (menu)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:37

S27	3	(((((ad<"19990913").ad. and download and (video near10 (sales distribution distribute rent rental))) and ((subscriber subscribe) and (device))) and ((subscriber subscription) near10 (identifier identification ID))) and (internet or web or URL) and (menu)) and (segment)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:42
S28	2	("5956716").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/12 13:43
S29	2	("5956716").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2003/05/13 08:12
S30	4	("6286002" "6119101").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 11:39
S31	1255	intelligent adj2 agent	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:05
S32	61	(intelligent adj2 agent) and (@pd<"19980122").pd.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:52
S33	45	((intelligent adj2 agent) and (@pd<"19980122").pd.) and (intelligent adj1 agent)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:27
S34	1	((intelligent adj2 agent) and (@pd<"19980122").pd.) and (intelligent adj1 agent)) and (peckover).in.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:37
S35	29	((intelligent adj2 agent) and (@pd<"19980122").pd.) and (intelligent adj1 agent)) and (queries requests inquiry inquiries)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:28

S36	6	(((((intelligent adj2 agent) and (@pd<"19980122").pd.) and (intelligent adj1 agent)) and (queries requests inquiry inquiries)) and (recommend recommendations suggestions suggest))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:28
S37	3	("9726612").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:37
S38	61	(@pd<"19980122").pd. and (intelligent adj2 agent)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:54
S39	2	((@pd<"19980122").pd. and (intelligent adj2 agent)) and (firewall)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2003/05/13 12:54
S40	2	("5701451").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 08:46
S41	180	(@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or feedback)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 09:24
S42	104	((@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or feedback)) and (accept or reject)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 09:29
S43	95	(((@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or feedback)) and (accept or reject)) and (trend or past or historical or future)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 09:45
S44	15	(((((@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or feedback)) and (accept or reject)) and (trend or past or historical or future)) and (award or incentive or reward)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 09:45

S45	1	(((((@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or feedback)) and (accept or reject)) and (trend or past or historical or future)) and (award or incentive or reward)) and (not feedback)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 09:48
S46	2	(((((@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or questionnaire)) and (accept or reject)) and (trend or past or historical or future)) and (award or incentive or reward)) and (not feedback)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 09:48
S47	8	(((((@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or questionnaire)) and (accept or reject)) and (trend or past or historical or future)) and (award or incentive or reward)) and (advertisement or offer)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 09:53
S48	8	(peckover).in.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/24 14:06
S49	2	("5,701,451").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/23 10:02

S50	19	(((((ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or questionnaire)) and (accept or reject)) and (trend or past or historical or future)) and (award or incentive or reward)) and (not feedback)) or ((((((ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and (protect or hide or firewall or fire\$1wall) and (survey or questionnaire)) and (accept or reject)) and (trend or past or historical or future)) and (award or incentive or reward)) and (advertisement or offer)) or ((peckover).in.) or ("5,701,451").pn.)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/24 08:39
S53	5	(@ad<"19960117").ad. and ((demographic or personal) near3 (information or data)) and ((trend or past or historical or future) near5 (sales or shopping)) and (award or incentive or reward) and (advertisement or offer)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:09
S54	31	(filepp).in.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 09:09
S55	4	(peckover).in. and (market\$4)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/24 14:06
S56	2	("5347632").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 09:09
S57	165	(@ad<"19960117").ad. and ((compar\$5 or rank\$4) near5 (shopping or buying))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:38
S58	42	((@ad<"19960117").ad. and ((compar\$5 or rank\$4) near5 (shopping or buying))) and (internet or web or network)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:12

S59	12	((@ad<"19960117").ad. and ((compar\$5 or rank\$4) near5 (shopping or buying))) and (internet or web or network)) and ("705"/\$\$. ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:20
S61	1	((@ad<"19960117").ad. and ((compar\$5 or rank\$4) near5 (shopping or buying))) and (internet or web or network)) and ("705"/\$\$. ccls. and gateway	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:20
S62	11592	(@ad<"19960117").ad. and ((advertis\$5 or offer\$4) and (reject\$4 or refus\$3))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:38
S63	347	(@ad<"19960117").ad. and ("705"/\$\$.ccls. and ((advertis\$5 or offer\$4) and (reject\$4 or refus\$3))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:39
S64	72	(@ad<"19960117").ad. and ("705"/\$\$.ccls. and ((advertis\$5 or offer\$4) and (reject\$4 or refus\$3)) and (reward\$3 or incentiv\$3)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:39
S65	55	((@ad<"19960117").ad. and ("705"/\$\$.ccls. and ((advertis\$5 or offer\$4) and (reject\$4 or refus\$3)) and (reward\$3 or incentiv\$3)) and (network or gateway)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/06/25 10:40
S66	34	((@ad<"19960117").ad. and ("705"/\$\$.ccls. and ((advertis\$5 or offer\$4) and (reject\$4 or refus\$3)) and (reward\$3 or incentiv\$3)) and ((network or gateway) and (coupon or discount))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 14:15
S67	432	(@ad<"19960117").ad. and ("705"/\$\$.ccls. and (survey or questionnaire or advertis\$4) and (advertis\$5 or offer\$4 or incentive)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 14:16
S68	98	((@ad<"19960117").ad. and ("705"/\$\$.ccls. and (survey or questionnaire or advertis\$4) and (advertis\$5 or offer\$4 or incentive)) and (demographic)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 14:22
S69	13	((@ad<"19960117").ad. and ("705"/\$\$.ccls. and (survey or questionnaire or advertis\$4) and (advertis\$5 or offer\$4 or incentive)) and (demographic) and (reject\$5) and (reason or explain or explanation)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 14:20

S70	13	(@ad<"19960117").ad. and ("705"/\$\$.ccls. and (survey or questionnaire or advertis\$4) and (advertis\$5 or reward\$4 offer\$4 or incentive) and (demographic) and (reject\$5) and (reason or explain or explanation))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 14:24
S71	3	(@ad<"19960117").ad. and ("705"/\$\$.ccls. and (survey or questionnaire or advertis\$4) and (advertis\$5 or reward\$4 offer\$4 or incentive) and (demographic) and ((reject\$5) near20 (reason or explain or explanation))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/06/25 14:24
S72	2322	(@ad<"19960117").ad. and (advert\$8 same (match\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/01/10 10:29
S73	2322	(@ad<"19960117").ad. and (advert\$8 same (match\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:29
S74	27112	(@ad<"19960117").ad. and ((advert\$8 coupon\$3 promotion announcement commercial notice) same (match\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:30
S75	46682	(@ad<"19960117").ad. and ((advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:31
S76	855	((("705"/\$\$.ccls. or ("709"/\$\$.ccls. or ("345"/\$\$.ccls.) and (@ad<"19960117").ad. and ((advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:45
S77	2376	((("705"/\$\$.ccls. or ("709"/\$\$.ccls. or ("345"/\$\$.ccls.) and (@ad<"19960117").ad. and (classifi\$5 (advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 track\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:51

S78	20	((("705"/\$).ccls. or ("709"/\$).ccls. or ("345"/\$).ccls.) and (@ad<"19960117").ad. and (classifi\$5 advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 track\$5 simliar\$4 alike different\$5)) and (classified near5 advertis\$4)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:47
S79	24	(classified near5 advertis\$4).ti.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:47
S80	20	((("705"/\$).ccls. or ("709"/\$).ccls. or ("345"/\$).ccls.) and (@ad<"19960117").ad. and (classified near5 advertis\$4)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:48
S81	2	((("705"/\$).ccls. or ("709"/\$).ccls. or ("345"/\$).ccls.) and (@ad<"19960117").ad. and (classified near5 advertis\$4).ti.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:48
S82	1220	((("705"/\$).ccls. or ("709"/\$).ccls. or ("345"/\$).ccls.) and (@ad<"19960117").ad. and ((classifi\$5 advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 track\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 11:02
S83	0	S82 and ((blind or shadow\$3) near5 advertisement)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:53
S84	1	S82 and ((blind or shadow\$3) same advertisement)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:53
S85	4064	((("705"/\$).ccls. or ("345"/\$).ccls.) and (@ad<"19960117").ad. and ((classifi\$5 advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (advert\$5 match\$5 track\$5 simliar\$4 alike different\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 10:59
S86	19	(shadow near5 advert\$5)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 11:00

S87	117	((hid\$8 or invisible) near5 advert\$5)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 11:00
S88	8	(705/14).ccls. and ((hid\$8 or invisible) near5 advert\$5)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 11:01
S89	17	((("705"/\$\$.ccls. or ("709"/\$\$.ccls. or ("345"/\$\$.ccls.) and (@ad<"19960117").ad. and ((classifi\$5 advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 track\$5 simliar\$4 alike different\$5)) and (want\$4 near5 advert\$8)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 12:00
S90	1	((("705"/\$\$.ccls. or ("709"/\$\$.ccls. or ("345"/\$\$.ccls.) and (@ad<"19960117").ad. and ((classifi\$5 advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 track\$5 simliar\$4 alike different\$5)) and (want\$4 near5 advert\$8) and ((hid\$5 blind or shadow or invisible) near5 (advertis\$6))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 12:05
S91	1	((("705"/\$\$.ccls. or ("709"/\$\$.ccls. or ("345"/\$\$.ccls.) and (@ad<"19960117").ad. and ((classifi\$5 advert\$8 coupon\$3 promot\$5 announc\$5 commercial notic\$4) same (match\$5 track\$5 simliar\$4 alike different\$5)) and (want\$4 near5 advert\$8) and ((hid\$5 blind or shadow or invisible) near3 (advertis\$6))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/01/10 12:05
S92	2	("5752238").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/10/04 15:03
S93	0	("5752238").pn. and (customer with offer\$5)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/10/04 15:03
S94	0	("5752238").pn. and (customer with offer\$5)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:04

S95	0	("5752238").pn. and (customer with (inquir\$5 offer\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:04
S96	0	("5752238").pn. and (consumer with (inquir\$5 offer\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:04
S97	0	("5752238").pn. and (consumer with (request\$5 inquir\$5 offer\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:05
S98	0	("5752238").pn. and (consumer with (request\$5 inquir\$5 offer\$5 post\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:15
S99	2	("5752238").pn. and (consumer with (advert\$8 request\$5 inquir\$5 offer\$5 post\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:20
S10 0	2	("5752238").pn. and (user with (advert\$8 request\$5 inquir\$5 offer\$5 post\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:30
S10 1	1	("5752238").pn. and (user with creat\$5 with (advert\$8 request\$5 inquir\$5 offer\$5 post\$5))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/04 15:30
S10 2	923	(smith near3 jeffrey).xp.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/10/13 12:55
S10 3	689	(smith near3 a near2 jeffrey).xp.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/13 12:56
S10 4	374	(smith near3 a near2 jeffrey).xa.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/13 12:56

S10 5	2	("20030154137").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 10:52
S10 6	345	(CARROLL near5 (JEREMY JOHN))).in. (SEABORNE near5 (ANDREW FRANKLIN)).in. (BATTLE near5 (STEVEN ANDREW))).in.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 10:53
S10 7	339	(CARROLL near5 (JEREMY JOHN))).in. (SEABORNE near5 (ANDREW FRANKLIN)).in. (BATTLE near5 (STEVEN ANDREW))).in. and IDL	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 10:54
S10 8	339	(CARROLL near5 (JEREMY JOHN))).in. (SEABORNE near5 (ANDREW FRANKLIN)).in. (BATTLE near5 (STEVEN ANDREW))).in. and IDL and protocol	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 10:54
S10 9	1	((CARROLL near5 (JEREMY JOHN))). in. (SEABORNE near5 (ANDREW FRANKLIN)).in. (BATTLE near5 (STEVEN ANDREW))).in.) and IDL and protocol	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 10:54
S11 0	22	((CARROLL near5 (JEREMY JOHN))). in. (SEABORNE near5 (ANDREW FRANKLIN)).in. (BATTLE near5 (STEVEN ANDREW))).in.) and protocol	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 10:54
S11 1	13	((CARROLL near5 (JEREMY JOHN))). in. (SEABORNE near5 (ANDREW FRANKLIN)).in. (BATTLE near5 (STEVEN ANDREW))).in.) and (contract agreement)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 10:56
S11 2	7	((CARROLL near5 (JEREMY JOHN))). in. (SEABORNE near5 (ANDREW FRANKLIN)).in. (BATTLE near5 (STEVEN ANDREW))).in.) and (contract agreement) and message	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 12:10
S11 3	0	(@ad<"20020213").ad. and (contract\$5 negotiat\$5) and (E\$1speak) and (IDL)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 12:12
S11 4	17	(@ad<"20020213").ad. and (contract\$5 negotiat\$5) and (E\$1speak)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/10/14 12:12

A Service Framework Specification for dynamic e-services interaction

Ravi Balakrishnan
Product marketing Manager, Espeak Division, Hewlett Packard Co.
Ravi_balakrishnan@hp.com

Abstract: This paper presents an overview of the concepts involved in implementing a totally inter-operable ecosystem of e-services encompassing both the business-to-business and business-to-consumer domains. These ecosystems integrate such things as supply chain automation, e-procurement, collaborative e-commerce, trading exchanges and personalized business-to-consumer interactions.

This paper details the e-speak Service Framework Specification (SFS) and its core API that form the foundation for intelligent interaction amongst e-services in the ecosystem defined above. Refer to Fig-1 for a detailed view of an e-speak ecosystem. This paper also addresses some of the cardinal elements of a truly inter-operable ecosystem such as compliance with XML based business process schemas defined by RosettaNet [2] and CommerceNet [3]. Note: The term e-service is widely used to refer any entity that offers a service over the Internet.

1.0 Introduction - Current Business

Situation: The Internet service economy is here and booming and particularly the B-2-B sector is growing at an unprecedented pace. The number of sites providing B2B or even B2C services has grown into hundreds of thousands. Aggressive consolidation and ruthless

competition has begun. However, in the rush to reach the market many service providers, service consumers, and financial intermediaries and trading exchange operators have become locked into proprietary protocols and standards with no possibility of inter-operating with one other. Worse still many of these service sites and vendors are locked into customized, non-extensible, monolithic business models and solutions.

In most present-day trading exchanges, once the supplier and buyer establish the relationship, the exchange loses its value. The reason is these exchanges do not provide value-added services. Compounded with the inter-operability problem listed above it is a true jungle out there. Analysts estimate that the current number of 600 or so trading exchanges would likely multiply to tens of thousands in the next few years. Even leading exchange operators do not provide features such as dynamic supplier discovery and inter-operability with other exchanges. But buyers and sellers wish to have flexibility in which exchanges they work with. Lock-in is a dreaded feature.

As an example, consider a case in which a company regularly purchases office supplies through its B2B trading exchange. If an order cannot be filled because a particular item is not in stock, the buyer becomes stuck without the possibility of discovering another supplier. Restocking the necessary item could take days, and the buyer would be forced to wait until the supplier was ready to fill its order. If suppliers were modeled as independent interoperable

102 (a)

services, however, the buyer would be able to automatically discover alternate vendors capable of delivering the same product, and could make an intelligent decision as to which new vendor it should buy from.

Or consider another case in which a small online business sells software to its customers worldwide. To compute the appropriate country-specific tax, and verify whether or not it is legal to sell its software to customers from a particular country, the e-business would have to incorporate the entire infrastructure into its business model, either by building the knowledge into its own site or by providing customized links to other sites or software that does contain this knowledge. It would be infeasible for a small company to either to build up such a knowledge base by itself. Customize all necessary links to other company's software might be infeasible for a small company as well.

E-speak Ecosystem

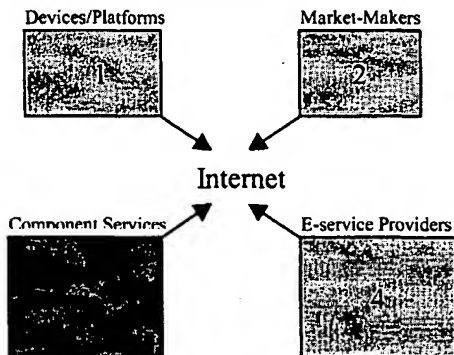


Figure-1. E-speak ecosystem

1. PC's PDA's and other devices
2. Various Market-Makers such as Gaming or Travel
3. Components such as billing/payment E-services
4. Various E-service providers such as Home Insurance, Title, or Real Estate E-services.

1.1 How e-speak SFS solves aforesaid problems: The E-Speak Service Framework specification addresses these problems by introducing a uniform service model for digitally capturing and representing individual pieces of business functionality. It defines both the technology level interactions between such business functionalities (how they address, send messages to, and authenticate each other), and unifies the formation of business relationships through negotiation, contract formation and business process integration. This common business and technology level interaction framework supports, co-habits with, and builds upon, several emerging industry standards, such as BizTalk [8], CBL, RosettaNet PIP [2] and IBM's tpaML [4]. By defining a uniform, non-proprietary, and extensible means for interaction between Internet services, business collaboration in the new economy through direct interaction, auction sites, exchanges or aggregators will become easier and less chaotic.

1.2 What is the E-speak Platform: E-speak is an open software platform designed specifically for the development, deployment and intelligent interaction of e-services. The platform consists of the e-speak SFS, e-speak Service Engine and the e-speak Trading Communities Edition. The e-speak SFS defines standards that allow e-services to dynamically discover and negotiate with each other and compose themselves into more complex services.

The E-speak Service Engine is a high-performance software implementation of the E-speak SFS. It is an open software platform designed specifically for the development, deployment, composition and intelligent interaction of Internet e-services. It is the preferred (but not mandated) implementation platform for supporting the service framework specification. The e-speak service engine provides the implementation abstractions

necessary for supporting the dynamic discovery of e-services, the ability to compose multiple e-services into higher-level e-services, multi-party negotiation algorithms, and the ability to manage service transactions. Additionally, the e-speak service engine implements a role-based security infrastructure, and facilitates dynamic yet safe traversal through firewalls. The e-speak Trading Communities Edition offers pre-integrated modules that reduce the time and effort it takes to build an online community.

1.3 How the SFS defines E-services

Interactions: The e-speak SFS defines standards that allow interactions among e-services. It provides a detailed framework in which Internet B2B e-services can discover each other, negotiate according to user-defined criteria, and reach agreement on product, pricing and delivery terms. Companies that develop e-services that comply with the e-speak SFS empower those applications to interact with other e-speak-enabled e-services to deliver even greater value to customers. Sites running the e-speak engine can automatically discover other e-speak sites and dynamically create compound e-services that address complex business needs.

The SFS is more powerful than traditional software component frameworks such as Corba, EJB [5]. It provides support for expressing higher-level business abstractions, negotiations, dynamic binding, interaction during execution and a loosely-coupled asynchronous messaging based on an open XML document exchange model. E-services that conform to the SFS can be deployed using the e-speak Engine, which acts as the intelligent platform that manages and monitors access to e-services, community portals and trading communities.

The e-speak SFS enables interoperability across almost any hardware, software, and operating environment by defining a common interaction model that supports the major emerging B2B e-

commerce standards. Using a common means of defining and implementing Internet e-services, companies can deploy e-services that can collaborate across organizational and market boundaries. Market makers, aggregators and auction services can freely interact in a dynamic, yet secure open services marketplace. In such a dynamic marketplace, the SFS adds value in the form of cross-platform interaction, componentization (ability to build small, medium and large exchanges), ease of configuration, diverse business models and end-to-end service provisioning.

1.4 Business Purpose of the SFS: The SFS defines an open XML-based, non-proprietary standard for defining interactions between e-services at both a technology and business level. It builds upon existing and emerging industry standards and attempts to provide a framework to easily integrate existing innovations in this space. Innovation is attempted only where absolutely necessary. This is an attempt to bring together several independently focused but innovative standards and protocol specification work that currently exists in the B2B space.

2.0 SFS Technology Overview

2.1 Why do we need a service framework:

Traditional software component frameworks provide an infrastructure for enterprise-level distributed application development. However, such traditional frameworks are not suitable for deploying services across the Internet due to lack of support for higher-level business abstractions, limited support for dynamic binding, and use of a network-object interaction model. The e-speak service framework solves these three problems by using a document-exchange programming-model and defining a set of service-level interaction APIs in the form of XML document schemas with associated semantics. These documents contain business-level information that allow services to

introspect one another, negotiate, establish contract-based relationships, monitor the progress of each transaction, and, if necessary, integrate business processes. The actual information exchange between e-services is service-specific.

2.2 What is an e-service: An e-service is any application, written potentially in any language, that conforms to a set of properties (meta-data) and can receive and respond to messages as defined by its service specification. The service specification is a set of two XML documents the first of which describes the interaction interface(s) of a service; i.e. the documents it accepts, the documents it returns, and the protocols it uses for carrying out its interactions. The second document of the service specification describes the meta-data or properties; ie: details about what the service does, service provider information, information about where related service information may be found and so on. How the service specification is exchanged between services is not important - it may be published at a well-known location, exchanged out-of-band, or provided by the service via introspection (described in this paper).

Conformance to the SFS implies that the service can exchange documents with other services according to the protocols and schemas laid out in its specification. Note that conformance to a valid specification does not necessarily mean that an e-speak service can participate in any given marketplace. Market participation may also require conformance to rules specified by the market-maker.

Services talk to each other through *conversations*. Each service can support one or more conversations. A conversation represents a set of *interactions*, where each interaction comprises of a request (a set of documents) being sent from one service to another and a

response (another set of documents) being received. The documents that are exchanged as part of an interaction consist of various *Information Elements* [Eco]. The notion of conversations, interactions and Information Elements have been borrowed from the Eco framework [3], and roughly correspond to sessions, method-invocations and objects in object-oriented frameworks.

2.3 SFS Architecture

Broadly speaking, the service framework architecture tries to answer the following three questions.

- **Service Interaction:** How should e-services interact over the open Internet?
- **Service Model:** What does a service look like?
- **Service Provisioning:** How should services be provisioned?

Service Interaction: The service interaction model primarily deals with messaging between e-services. In e-speak SFS, messages are abstract representations of interactions and conversations that define the schemas for a message and the contextual information necessary to communicate the abstraction state.

Service Model: Using introspection, an e-service can reveal information about its properties and the conversations it supports. The definition of the schemas for Property Sheets and Conversation Process Definitions detail the information that is actually passed back and forth between say, a client (buyer) and a service provider (seller) in this environment. This information decides the contents and sequence of interactions between a buyer and a seller.

Service Provisioning: Services are provisioned by buyers and sellers advertising (or registering

offers) through a matchmaker, which takes on the role of effecting introductions upon a lookup request. A sequence of negotiations through the exchange of negotiated offer documents follows. At the end of this phase a contract document exists, which may be maintained by a market-maker or by the participants themselves. These protocols setup the interaction between multiple interacting e-services.

2.3.1 Components of SFS: In line with the three models described above, the SFS consists of the following components:

- A. Messaging specification (Service interaction model)
- B. Introspection specification, Security specification, Application Management and Configuration specification (Service model)
- C. Dynamic negotiation specification, Matchmaking specification, Service composition specification and a Deployment specification

These specifications, along with the SFS programmer's guide, the SFS client libraries, and the e-speak service engine, constitute the full suite of tools for building inter-operable, end-to-end solutions in the B2B and B2C space.

A brief description of some of the SFS specification components is presented below.

2.3.2 Service Introspection Specification: The basic concept of introspection is quite well known. The Java programming language, for example, defines introspection as a mechanism that software components can use to expose their interfaces for dynamic code extension and manipulation. The service framework extends this traditional concept to include an intelligent sequencing engine capable of discovering not only the interfaces, but also the *sequence of interactions* within the interfaces. Many present day B2B interactions typically operate according

to pre-arranged document-exchange protocols, with a pre-agreed set of documents. The suppliers and partners involved in each transaction are well known, and operate according to proprietary protocols and standards. There is no provision for, or possibility of, dynamic interaction with unknown partners. E-speak service introspection solves this inter-operability problem by presenting a new way for suppliers, partners, and customers to interact in a dynamic, intelligent manner.

To dynamically interact with the services it finds on the network, a service must first discover:

1. What kind of service is being offered
2. How to interact with the service

Every e-speak service must provide two interface documents: a *service property definition* and a *service conversation process definition*. A service property definition describes general service information, service provider information, service registration information, service conversation definition location, service credentials, and legal terms. It describes *what* functionality the service can provide. The service conversation process definition, on the other hand, describes service interaction rules, such as the type of request document the service will accept, the type of response document the service will produce, and the conversation definition language used to dictate state and transaction information. It describes *how* a client interacts with the service.

Figure-2 below describes the introspection interfaces of a service.

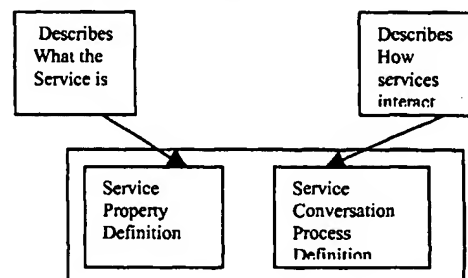


Fig-2: A single service

A sample XML snippet of a Service Property Sheet Request is shown below.

```
<?xml version="1.0">
  <Espeak
    version="1.0" xmlns="urn:schemas.w3c.org:espeak:e-speak-1.0"> [7]
    <Header>
      <From>

      <!--From
information goes here --!>
      </From>
      <To>

      <!--To
information goes here --!>
      </To>
    </Header>
    <Body>
    <GetServiceProperty>
      <!--Additional parameters
if any --!>
    </GetServiceProperty>
    </Body>
  </Espeak>
```

2.3.3 Messaging Specification: This section describes the information exchanged between service-tiers. The following description demonstrates how a document passes through the different layers. A service developer calls a Java-library that constructs an XML document and sends it to the service-tier, along with some routing information about who the document has to be sent to. The service-tier packages up the document into an **<Espeak>** document consisting of a **<Header>** and a **<Body>**. The Header contains information sent by one service-tier to the other telling it where to dispatch the document, and the context in which the document should be interpreted.

A sample XML messaging schema is illustrated below.

<ES:Header>

```
<ES:Delivery>
  <ES:From>
https://acme.com#PurchaseConv </ES:From>
</ES:Delivery>

<ES:Context>
  <ES:Conversation
    name="PurchaseConversation.cbl" |
    id = "3214132414"
    remoteld = "245425452511"
  />
</ES:Context>

<ES:Manifest> Purchase Order from
Acme Corporation </ES:Manifest>
</ES:Header>
```

The **<Body>** of an E-speak message contains content that is exchanged between applications and is not changed by the service-tier. Applications (and service-tiers) can also append other documents as attachments to the **<Body>**. These attachments need not be XML documents. Attachments allow applications to efficiently exchange binary data. It also allows service-tiers to piggyback other documents with the primary document contained in the **<Body>**. Routing of messages in the service framework is based on the notions of connections, sessions and conversations. *Connections* are communications channels used by the communication tier to connect to a routing framework. A connection allows a communication-tier to dispatch a message into the routing framework and allows a recipient that is connected to the same routing framework to receive the message.

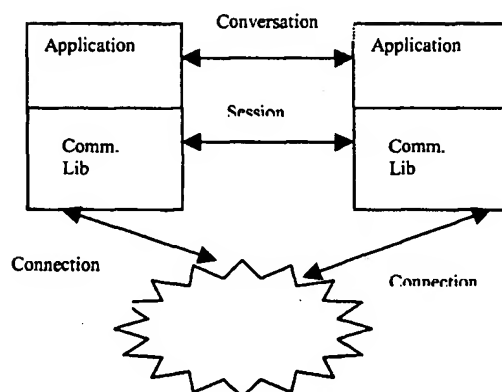
Sessions are end-to-end communication channels between two services. Sessions have state that has a lifetime greater than that of a single message. This state allows an end-point to associate certain properties with messages that arrives in a given session. The state may be as simple as the information needed to implement a request-reply protocol (e.g. an HTTP session), or may be as complex as the

state needed to implement a secure-session (e.g. HTTPs or SLS). In order to set-up a session, the two end-points need to have connections to a common routing framework. However, multiple sessions can share the same connection. The lifetime of a session does not necessarily depend on the lifetimes of its underlying connections.

Conversations are stateful communication channels whose lifetime is independent of the underlying communication framework and is controlled solely by the application. Conversations are initiated by the application and may last for a very long time. Hence conversational state is backed by persistent memory. A conversation depends on underlying sessions to communicate, but it may use multiple sessions concurrently, share the same session with other conversations, or may even depend on sessions in multiple routing frameworks (SMTP, HTTP) simultaneously.

Refer to **Figure-3** below for the relationships among conversations, sessions and connections.

Figure-3. SFS messaging abstractions



2.3.4 Management Specification: This specification defines protocols and commands for controlling and monitoring e-services. It incorporates ARM (Application Response Measurement) standards, which provides for application programmers to instrument their code by inserting ARM library functions in appropriate transactions.

2.3.5 Matchmaking Specification: The matchmaker is where services that want to be dynamically discovered register themselves (i.e., their advertisements, or offers), and where services that want to dynamically find other services send their requests for matches. Essentially, the matchmaker itself is a service that participates in the services framework. The matching functionality provided by matchmakers has two possible flavors: simple and complex. It can vary from a simple catalog management system to a complex commodity exchange. In simple matching, the eventual buyer or consumer drives the matching process. For example, in the case of a simple catalog management system each entry in the catalog can be treated as an offer to sell the good or service described in the catalog. The client who is interested in buying goods browses the catalog and picks a supplier. Such a protocol is popular in a business to business scenario, where a company interested in purchasing some good creates a request for quotes (RFQ) that suppliers respond to.

In complex matching, both the supplier and the buyer have control over the matching process. Examples of complex matching processes include auctions and exchanges. In an auction, one end of the matching process is a single entity (the entity that initiated the auction), whereas in an exchange, the matching process is a many-to-many matching process.

In essence, matchmakers can be classified along the following dimensions:

- Kinds of offers they support (offers to buy, offers to sell, or both)
- Kinds of matching they support (simple matching, complex matching, or both)
- Kinds of interactions they support (on-line, off-line, or both)

2.3.6 Dynamic Negotiation Specification: The negotiation specification describes the protocol to be used for negotiating any document. Essentially, when the initiator of the first offer to negotiate makes a negotiation offer, the recipient outlines the list of negotiables in the NegotiableItems element of the offer. The recipient typically responds with a counter offer with the ability to add to the negotiable list. This first round determines the complete list of negotiables of the negotiation protocol.

2.3.7 SFS Summary: In summary, the SFS specifies the various components of an intelligent and dynamic e-services interaction model. Intelligent interaction is contained in the introspection, dynamic negotiation and service composition aspects while dynamic interaction is embedded in the discovery and lookup of e-services, dynamic negotiation and service composition features. The SFS transcends all competitive frameworks that are out there in the marketplace. A comparative analysis of E-speak SFS, IBM's tpa-ML, CommerceNet's Eco, RosettaNet and Microsoft's Biztalk frameworks is presented in Fig-4 below.

SFS	eCo	tpa-ML	Espeak	Rosetta Net	Biztalk
Service Spec.	X	X	X	X	X
Messaging Spec.	X	X	X	X	X
Deploy	NO	NO	X	NO	NO

ment Spec.					
Manag ement Spec.	X	X	X	NO	NO
Measur ement Spec.	NO	NO	X	NO	NO
Securit y Spec.	NO	X	X	X	X
Negoti ation Spec.	NO	NO	X	NO	NO
Introspe ction Spec.	X	NO	X	No	NO
Contra ct Spec.	NO	X	X	X	NO

Fig-4: Framework Comparison Table

In the above table, an 'X' means presence of a feature whereas 'NO' means absence.

The SFS is a flexible framework and can be configured into small, medium and large-scale business exchanges. The SFS is standards based and open protocol based and inter-operates with frameworks such as tpa-ML, Biztalk and standards such as eCo, RosettaNet.

The SFS is an XML based document exchange framework and supports multiple protocols in its communication (RMI, IIOP, MSMQ etc) and transport (http(s), ftp, SMTP etc) layers. The SFS supports MIME as a content encoding type and offers S/MIME and PKCS for data encryption and digital signatures. Custom encoding, security and transport settings may also be plugged directly into the SFS. This encoding architecture facilitates data integration to be independent of the network protocol used to communicate with trading partners.

The E-speak SFS transcends the other frameworks by offering the value proposition of intelligent interaction. The central theme around

intelligent interaction involves introspection, dynamic negotiation, dynamic composition and dynamic discovery. These powerful concepts drive a new generation of B-2-B e-commerce in which eco-systems of suppliers, resellers, distributors and buyers are formed dynamically. Buyers can discover new suppliers and negotiate on the fly to form new contractual relationships. E-services can be dynamically composed to form more powerful composite services. The SFS's support for dynamic pricing enables multi-party negotiation and provides support in setting up trading exchanges. The E-speak SFS adopts standards from RosettaNet and CommerceNet in its specification. For instance, SFS is RosettaNet PIP compliant. The SFS also supports RosettaNet standards in product classification schemes such as D&B (Dun and Bradstreet), UN/SPSC etc.

The SFS uses RosettaNet compliant business schemas in its conversations such as Purchase Order, Shipping notification, Order management and other supply chain applications.

The SFS is also Biztalk compliant in that Biztalk messages can be wrapped around e-speak messages and vice-versa.

3.0 Conclusion

3.1 Future Steps: This specification lays out initial constructs in the interaction between services, modeling of services and provisioning of service over the open Internet. These aspects will be extended in the future to allow for composition of services, notification and synchronization between services. Support for role-based security protocols will also be added. The specification is meant to garner interest and comments from parties interested in participating in the process of standardization.

32. Summary: This paper presents the problems faced by the online marketplace today and outlines a highly effective solution to address them. The main problems are lack of inter-operability/liquidity, complexity of configuration, lack of componentizability, and disconnects in end-to-end service provisioning. Future e-commerce landscape will be an even more complex mix of multiple vendor systems, with need for integration across many different business process systems. Proprietary, non-standards solutions will not scale. There will be a need for seamless transfer of suppliers and buyers across exchanges, demand for components that will offer just-in-time services such as billing, payment etc. The SFS addresses all of these and opens the possibility of an e-services vision that is standards based, scalable, extensible and inter-operable.

4.0 References

- [1] www.hp.com (SFS, A service framework specification for building an inter-operable e-services ecosystem, architected and developed by Hewlett Packard's Espeak division)
- [2] www.rosettanet.org (A standards based organization for developing XML business process schemas and contributed to the development of Partner Interface process PIP)
- [3] www.commercenet.org (A standards consortium for developing business process integration framework and XML business process schema standards)
- [4] www.ibm.com (tpaML, trading partner markup language, An electronic contract specification designed and developed by IBM)
- [5] www.java.sun.com (Enterprise Java Beans (EJB), a component based architecture designed and developed by Sun Microsystems Inc.)

[6] www.xml.com (An organization for developing open XML standards)

[7] www.w3c.org (worldwide web consortium, A standards body that contributed to the evolution of http, SOAP protocols)

[8] www.biztalk.org (Biztalk Schema Library, An initiative from Microsoft Corp. for producing standard XML schemas for business process flows/integration)

digital library

[DIGITAL LIBRARY HOME](#)[BROWSE BY TITLE](#)[BROWSE BY SUBJECT](#)[SEARCH](#)[LIBRARY/INSTITUTION
RESOURCES](#)[RESOURCES](#)[SUBSCRIPTION](#)[ABOUT THE DIGITAL LIBRARY](#)[Archive Page >> Table of Contents >> Abstract](#)Fourth International Enterprise Distributed Object
Computing Conference (EDOC'00) p. 28

A Service Framework Specification for Dynamic e-Services Interaction

R. Balakrishnan

Full Article Text:



PDF



BUY ARTICLE



IEEE XPLORE

DOI Bookmark:

<http://doi.ieeecomputersociety.org/10.1109/EDOC.2000.882341>

Abstract

This paper presents an overview of the concepts involved in implementing a totally inter-operable ecosystem of e-services encompassing both the business-to-business and business-to-consumer domains. These ecosystems integrate such things as supply chain automation, e-procurement, collaborative e-commerce, trading exchanges and personalized business-to-consumer interactions. This paper details the e-speak Service Framework Specification (SFS) and its core API that form the foundation for intelligent interaction amongst e-services in the ecosystem defined above. This paper also addresses some of the cardinal elements of a truly inter-operable ecosystem such as compliance with XML based business process schemas defined by RosettaNet and CommerceNet.

Additional Information

[Back to Top](#)

Index Terms- electronic commerce; service framework specification; dynamic e-services interaction; totally inter-operable ecosystem; business-to-business; business-to-

[Abstract](#)
[Abstract](#)
[Index T](#)
[Citation](#)

Free acc

☐ Abstra
☐ Select

Electron in to

☐ Acces
text a
☐ Down
of PDI
[Subscri](#)[Get a W](#)

consumer; ecosystems; supply chain automation; e-procurement; collaborative e-commerce; trading exchanges; e-speak Service Framework Specification; intelligent interaction; truly inter-operable ecosystem; XML based business process schemas

Citation: R. Balakrishnan. "A Service Framework Specification for Dynamic e-Services Interaction," *edoc*, p. 28, Fourth International Enterprise Distributed Object Computing Conference (EDOC'00), 2000.

Usage of this product signifies
your acceptance of the Terms
of Use.

This site and all contents
(unless otherwise noted) are
Copyright © 2000, IEEE, Inc.
All rights reserved.

E-speak E-xplained

Alan H. Karp
Hewlett-Packard Laboratories
Palo Alto, California

The Internet, World Wide Web, and Web browsers have brought us to the brink of a new way of doing business, and electronic economy, an *e-economy*, if you will. The picture has not been complete, though, preventing us from moving to the next level. E-speak provides the missing pieces. This report describes the various components of the e-economy and how e-speak fills the gaps.

1.0 Components of the E-economy

Buyers and sellers have been part of economic interaction since the earliest days of barter. The problem is how a buyer willing to purchase a product or service finds a seller willing to provide it. Advertising is one approach, but something more is needed for the e-economy, a *matchmaker*. The matchmaker connects those with an *offer to buy* with those with an *offer to sell*. Unlike static advertising, such as print and broadcast ads, the matchmaker deals with a truly dynamic environment, one in which offers come and go at a furious pace. The matchmaker also recognizes the symmetry between buyers and sellers, something missed in the advertising model. The e-economy is not just buyers bidding for products or services in an auction; it is also sellers actively seeking buyers in a reverse auction.

The job isn't done once the buyer and seller have been identified; they need to reach an agreement, *i.e., form a contract*. Contract formation can be quite simple in a face to face purchase of tangible goods. The buyer gives the seller cash, and the seller provides the merchandise. Indirect transactions, such as buying from a catalog, are not so simple. What if the buyer doesn't pay? What if the merchandise isn't shipped? Each of these cases involves fraud on the part of one of the parties. Even without fraud, there can be problems. What if the product isn't what the buyer expected? What if the merchandise is damaged or lost in shipping? What if the buyer's payment is not received (cash) or honored (check)?

A number of solutions have been tried. Cash on delivery (COD) puts trust in a third party to deliver the goods and forward payment. Credit card companies provide a similar role without being involved in the delivery of the goods. They guarantee payment to the seller and delivery of the promised product to the buyer. Without these guarantees, mail order sales would be a small fraction of what they are today. The COD service and credit card companies are primitive forms of the last component of the e-economy, that of *market maker*.

The market maker assists in the formation of contracts between buyers and sellers. There may be many buyers for a given product or service; there may be many sellers competing for a buyer's business. Each wants a contract on the most favorable terms. Forming the contract can be quite complicated, including not just price, but also things like quality, return policy, delivery time, *etc.* The market maker provides the means for negotiating these terms and conditions. In addition, the market maker monitors the transaction, guaranteeing delivery of the goods and payment. These components are shown in Figure 1. Often the roles of matchmaker and market mediator are played by the same party, a *broker*. Sometimes the broker buys from the seller and resells to the buyer, the traditional role of a broker.

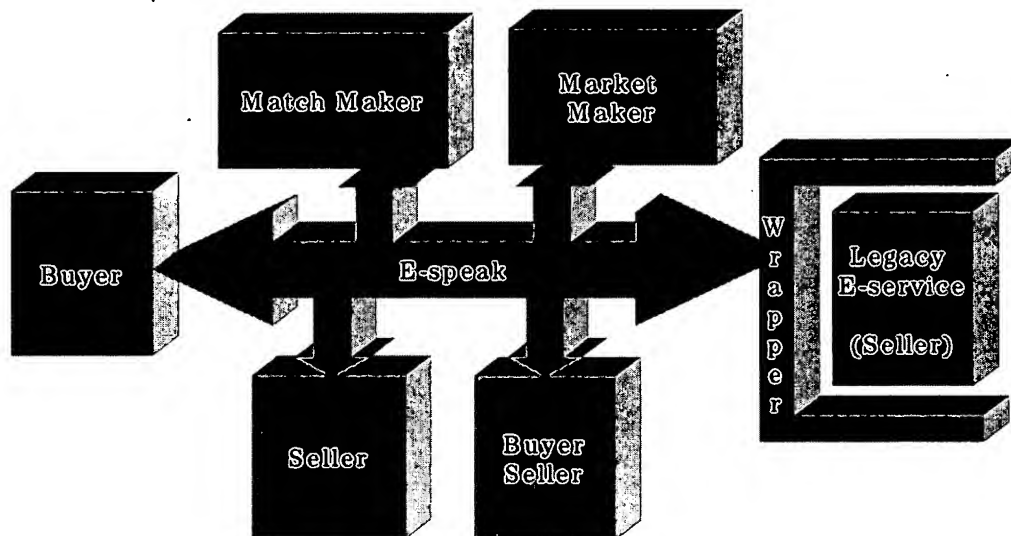


Figure 1. Components of the e-economy.

The problem today is that these components - buyers, sellers, matchmakers, market makers - have no framework in which to operate. Various projects and products have attempted to address parts of the problem. For example, Microsoft's Biztalk® and the eCo framework from CommerceNet address the problem of describing business logic, but they say nothing about discovery or security. HP's Praesidium® product deals with authentication and authorization, but says nothing about service discovery. Sun's Jini® addresses discovery and service invocation but says nothing about security or billing and manageability.

E-speak assists the creation, composition, mediation, virtualization, management, and accessing of Internet based services. It is an *open services platform* that lowers the barriers to creating new e-services. Each of the emphasized words is significant.

- *Open* - published interfaces, open source
- *Services* - designed for dynamic composition of e-services
- *Platform* - provides naming, discovery, management, and security

2.0 How E-speak Fits In

E-speak has a very modest goal - simply to change the way the world thinks about computing. Today, we think about computing as the hardware we buy and the software we install on it. E-speak is all about thinking of computing as a set of services that we enlist as we need them.

The essential difference is that when thinking about hardware plus software, we're always telling the computer how to do the job. "Go to that server. Use this printer. The paper I want is in the upper tray not the lower one." These are facts I may not know if I've just walked into someone else's office, and I want to print a file. If printing is a service, I say instead "I need a printout of this document, here, in color, with at least 600 dots per inch resolution, by 10 AM." I don't care how the job gets done. Maybe the print shop at the corner did the job and delivered the output.

We can see from this simple example, that e-speak's goal is to make it as simple, transparent, and safe to use an e-service as it is to access a web page using a browser. There is no need to know where the service is, how it is implemented, what kind of machine is used. I'm not putting myself at serious risk of attack or virus infection by using the service. However, adding computation to data delivery dramatically complicates the process of service creation. That's where e-speak comes in.

2.1 E-commerce, E-business, E-services

We've all heard these terms. Some pundits say that they're just marketing hype. Sun Microsystems grabbed "e-commerce"; IBM, "e-business"; all that was left for HP was "e-service." This perception is wrong; there are real differences as shown in Figure 2.

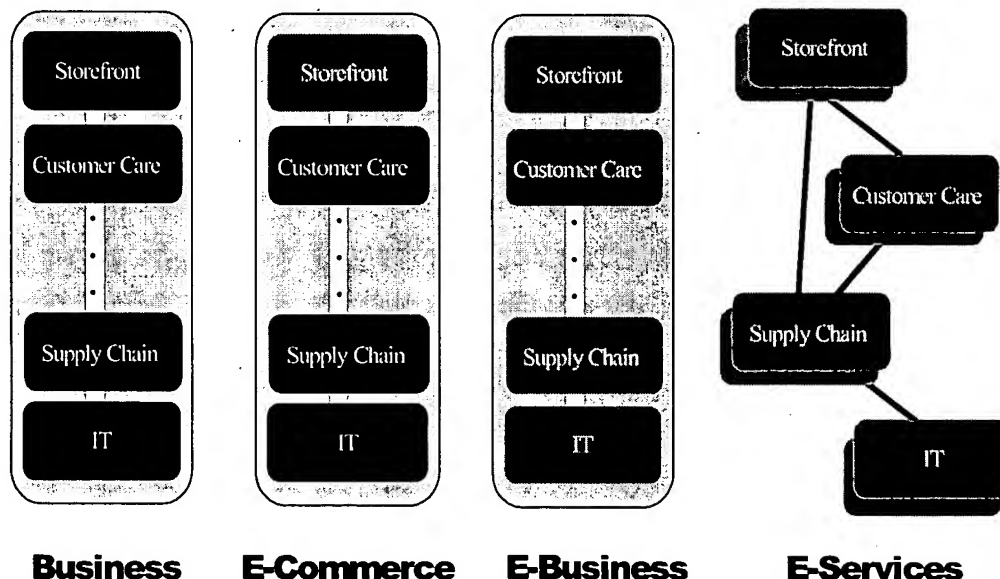


Figure 2. E-commerce, E-Business, and E-service comparison.

Each business today has a certain structure that constrains how data flows. For example, an order passes from the storefront through customer care, to credit management, to inventory, and finally to shipping. E-commerce puts the customer facing part of this process on the web. E-business puts the inward and supplier facing part of the business on the web. These are wonderful things to do; enormous efficiencies are gained. A large part of the productivity growth seen in recent years no doubt comes from this process.

However, neither e-commerce nor e-business changes the fundamental structure of the business to enable it to react to changes in Internet time. That's where e-services come in. With e-services you think of your business as a set of independent services that you enlist as you need them. Now, you don't have to wait for the next corporate reorganization to fix a broken process. Your processes adapt themselves to the need at hand.

There's another advantage to the e-services way of thinking. Let's say that there's one of these services you don't do very well. Out source it! Find someone out there who does the job better than you and is offering it as a service. In other words, out source your core incompetencies. Since your business is already structured to deal with independent services, there will be no disruption.

What if you don't find anyone who does a particular job better than you're doing it? Well, then you've got the best e-service available. Market it! Turn your core competencies into a revenue stream.

2.2 The Open Services Marketplace

Following this line of thought to its logical conclusion leads us to the idea of an open marketplace for such e-services. However, there's a problem. Look at the poor souls in Figure 3. The guy has a great idea for an e-service, but before he can offer it, he has to solve a number of problems, *e.g.*, advertising, billing, security, *etc.* The gal could make more money if she used this service; her customers could have a better experience. However, before using the e-service, she, too, has a number of problems to solve, *e.g.*, finding the service, monitoring its use, security of her system, *etc.*

The problem is that today, they each solve these problems independently. It's a terrible waste of time and money. Money today isn't a big deal, at least here in Silicon Valley, but time to market is critical. However, there's an even more serious problem. Because they've solved these problems independently, they've solved them slightly differently, making it difficult for them to interoperate.

E-speak is a layer of software running on collection of equipment that provides a set of mechanisms for solving this common set of problems. Because the mechanisms are the same for everyone, it's easier for the two parties to interoperate. E-speak also provides a language for specifying a variety of policies. "This service costs \$1.00 per use or \$50.00 per month." "Joe may use this service, but Sally may not." It is this combination of a platform providing the common set of mechanisms and the language for setting policy that makes e-speak a critical part of the open services marketplace. If this is all you understand about e-speak, you've captured the essence of what it does.

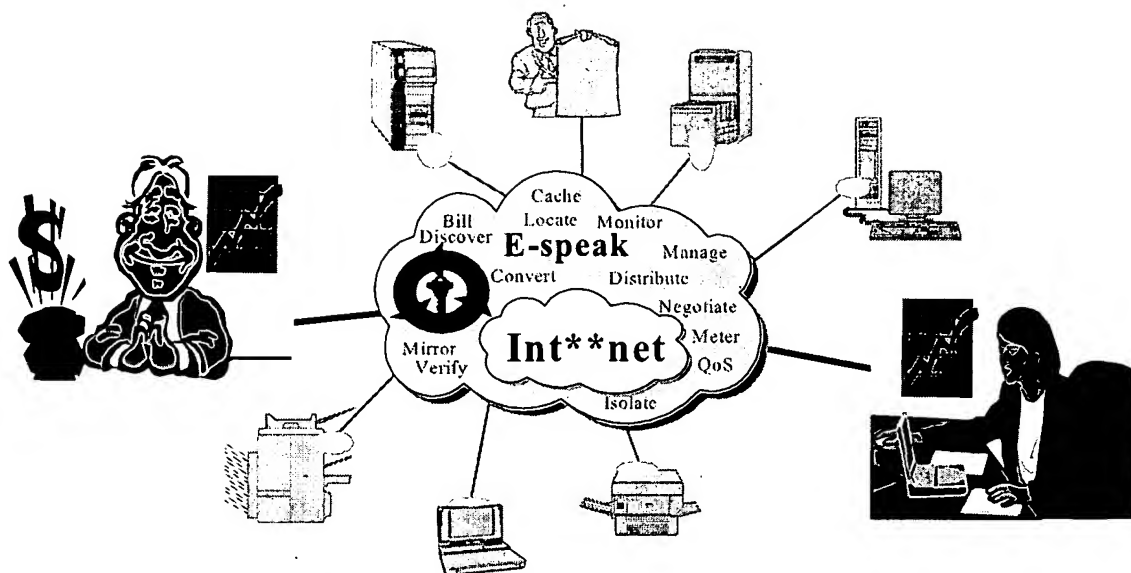


Figure 3. E-speak facilitates the open services marketplace.

2.3 Service Composition

Nobody knows how to make a pencil. One person makes the graphite; someone else, the eraser. A third party makes the little metal band that goes around the top. Very interesting, but what's this got to do with e-speak? Simple. E-speak encourages the same kind of service composition that goes into the making of a pencil.

Say that I want to go into a business with several components, none of which I know how to do. Integrated financial management is one good example. Such a service might include payroll, tax preparation, and cash management. With e-speak, I can easily enter this business as long as each piece is available to me as an e-speak service.

Figure 4 illustrates the process. First, I describe the business logic of my new service. First do the payroll, then calculate the taxes. Finally, in the unlikely event that any cash is left over after taxes, figure out how best to manage it. I'll also describe my policies, such as billing, authentication, and the like. Once that's done, I can advertise the service on the Internet or the Intranet. E-speak doesn't have to be everywhere; you can benefit if you just use it to streamline your intra-enterprise applications.

Once the service has been advertised, a customer can discover it and use it. There's something significant about this process. You'll notice that the service doesn't have to be configured to run specifically in the customer's environment. That's because I've told the system what job I want done, not how to do it. I haven't said "Use this specific computer." Instead, I said "Use a computer with these attributes." As long as the customer environment has a computer with the right

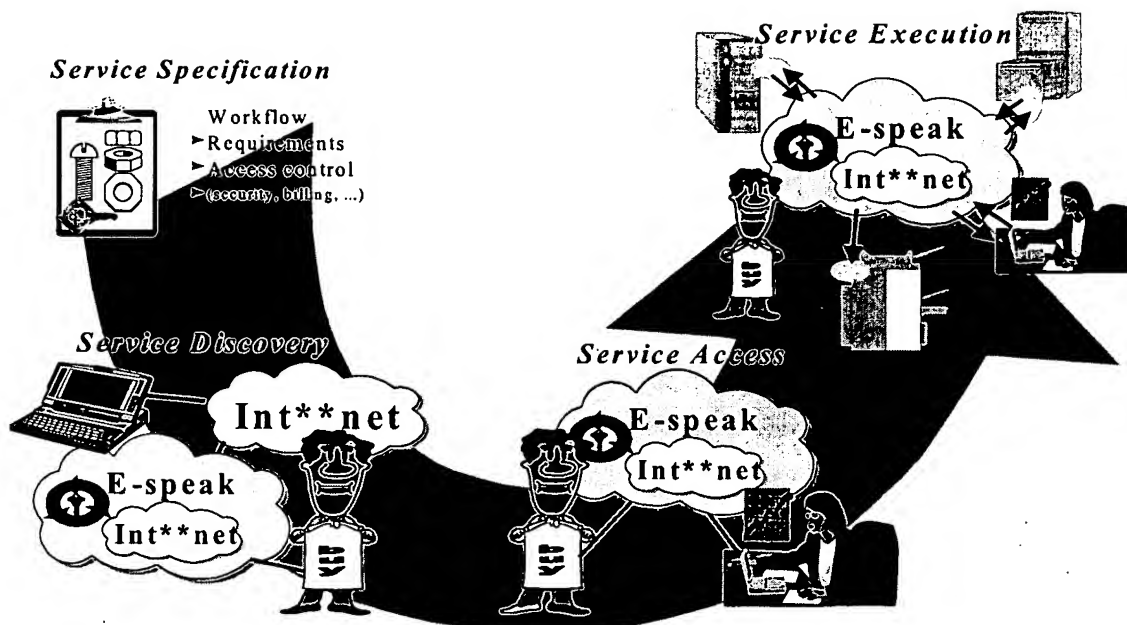


Figure 4. E-speak service composition.

attributes, the job runs correctly. No need for me to come configure the software for each installation; no need for the customer to run Install Shield® and go through six reboots.

What if the customer doesn't have a computer with the right attributes in her environment? That's the real beauty of thinking e-speak. It doesn't matter. As long as someone is willing to provide a computer that can do the job, my service will still run correctly. I didn't have to do anything special to make my application work; the e-speak platform took care of it for me.

3.0 The E-speak Stack

Other products focused on the e-economy are all point solutions, each addressing a particular aspect of the e-economy. E-speak, on the other hand, addresses the full set of key problems - naming, discovery, management, security, programming, and interoperability - in a single, coherent architecture. E-speak was not designed to solve a particular problem; it was always intended to be a general platform for doing the kinds of things needed to make the e-economy flourish.

E-speak is a complete platform as shown in Figure 5. One part implements the necessary mechanisms. Another provides the support to write applications. There is also a means to describe the service's business logic using one of a number of frameworks. Key to e-speak is the service bus that defines a way for services to interoperate. E-speak also provides a set of infrastructure services that provide the basic tools needed to bring an e-service to market.

At the bottom of the stack is the transport layer. The transport is not part of the e-speak architecture, only the interfaces to it are. Hence, e-speak today runs on TCP/IP, HTTP, and shared memory within a Java Virtual Machine. It should be simple to provide for other transports, such as

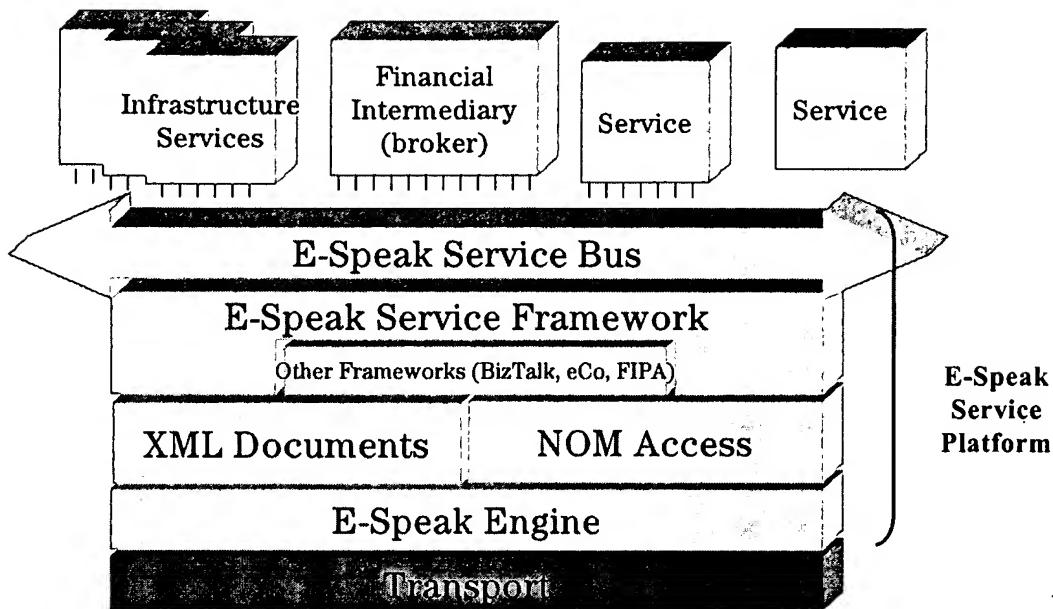


Figure 5. The e-speak stack.

wireless. At the top are the services; they are also not part of the e-speak architecture. However, any service that supports the interfaces defined by the E-speak Service Bus can be used as a building block of a higher level service.

The rest of the stack is defined by e-speak. The engine provides the operating platform on which the other parts ride. It handles the basic abstractions of naming, discovery, management, and security. The engine also mediates all requests, generating the proper management and billing events. Several programming models are built on this base, the main ones being direct messaging, document exchange, and network objects. E-speak does not mandate the way business logic is described. Instead, it provides a framework for using any available system.

The most important part of the stack, the part that makes dynamic composition of services a reality is the service bus. The bus itself doesn't exist as an entity any more than the World Wide Web is a physical entity. Instead, like the Web, the e-speak service bus consists of pieces of services that implement its interfaces. It exists so that any compliant service can be used as a building block of higher level services without a human being needing to read documentation and implement interfaces. It includes specifications for discovery, introspection of both interfaces and protocols, negotiation and contract formation, and service invocation. Hence, the service bus is a key enabler of dynamic service composition.

Today, I can go to your web site, read your user's guide, and start invoking your service. The problem is that this procedure is labor intensive and too slow to react to changes in real time. By supporting the service bus, you are assured that other services can find your service, negotiate and form a contract with it, learn how to use it, and invoke it, all without human intervention.

The e-speak platform and service bus specification reduce the barriers to producing e-services, but there are also infrastructure services that almost any business needs, things like billing, credit management, authentication, and authorization. If every provider of an e-service had to implement these services, the barriers to entering the e-economy would be quite high. In order to address this problem, the service bus specifies that there will be at least one instance of each of these infrastructure services available. HP is providing reference implementations that will be released to the open source community and is working with partners to provide commercial versions.

The service bus is defined in terms of a set of XML documents that can be communicated over the web on a universally available transport, HTTP. Thus, entering the e-economy can be as simple as filling in the blanks in a form.

4.0 History

E-speak was delivered to the world just before the turn of the millennium. Its origins go back almost 60 years, though, to a report by von Neumann describing a *computing utility* and to Vannevar Bush's Memex.

The modern history of e-speak began with Joel Birnbaum's 1982 paper on the "Information Utility". In that paper he used the word "Utility" in the sense of the electric utility; you don't think about it unless it's not there. You never think about electricity until you flip the switch and the lights *don't* come on. In addition, Birnbaum's "utility" was to be pervasive. Every computer, every printer, every automobile, every microwave oven, maybe even every light switch would be part of the utility infrastructure. It was quite an amazing vision given the state of the technology in 1982. The work on e-speak did not start from a formal requirements statement, but this vision had an important impact on its design.

There were other influences. In the mid 1980's, Bill Rozas, a graduate student at MIT at the time, was tired of having to use someone else's names, defaults, environments, even screen colors when using that person's computers. He argued for a *persona* that he could take with him anywhere in the world. In the late 1980's, I came to the rather obvious realization that it's easier to collaborate with someone on the same computer than with someone on another computer. My solution was to put everyone in the world on the same computer. Since one piece of hardware could never handle the job, the question became how to make all the computers in the world look like one computer.

In 1991, concerned about the complexity and expense caused by the rapid advances in technology, Rajiv Gupta started thinking about ways to reduce the total cost of ownership of computers and the associated infrastructure. He wanted to find a way to continue to use older machines even though they might not be able to run the latest applications. Also in the early 1990s, Arindam Banerji was trying to find a way to get a computer to deal with a new kind of thing, like an audio stream, without waiting for the next release of the operating system. His Ph.D. thesis showed how to allow a user to extend the capability of the operating system without compromising its security.

Each of these perspectives influenced the design of e-speak. As you read on, think about how the various features reflect these inputs.

We started the work that led to e-speak toward the end of 1995. By March of 1996, Bill Rozas and I had developed the beginnings of an architecture and a first prototype. Rajiv Gupta, at that time manager of HP Lab's Future Systems Department, would comment on what an interesting service we had just shown him. He often made suggestions, both on the architecture and the demonstrated uses of it. When we had implemented these ideas, he'd comment on how useful or profitable such a service would be. Over time, perhaps the six months from March to September of 1996, he formulated the idea of a world of e-services. Not just isolated e-services, but a world in which everything became an e-service. It was an interesting process to watch.

5.0 Why We Weren't There

Until e-speak, none of the perspectives presented above had been met. Why? One reason might be bad assumptions.

In the early days of computing, the 1950's and into the early 1960's, a computer was a big machine kept in a locked room attended to by specially trained people. Most users had limited interaction with the machine. Normally, you'd hand a deck of cards through a small window, and a couple of hours later you retrieved some paper that showed you'd forgotten a parenthesis in your Fortran program!

As time progressed, computing moved away from the locked room in the form of terminals, first teletypes, then other typewriter based devices, and finally cathode ray tubes. Programmers were freed from physically visiting the computer or waiting for a courier to do so. Unfortunately, the mainframe computers were expensive, and many felt that these precious resources were wasted dealing with all those keystrokes.

Once inexpensive (\$100,000) computers could be built, there was a better solution in the form of client-server computing. It is based on the idea of a small computer close to the user that handles the mundane tasks of input and output while the *real* work is done by the mainframe.

The problem was that we carried our assumptions forward as we rolled out the infrastructure. One such bad assumption was that all users were dealing with a shared pool of resources. After all, that's really what a mainframe computer is, a shared pool of resources. When problems caused by these bad assumptions became apparent, people attempted to find point solutions to address a particular symptom. Network File System[®] from Sun Microsystems is one such point solution.

As time went on, we evolved from the controlled, proprietary client-server world to the wide open Internet. Unfortunately, we carried our assumptions forward with us again, and again we are attempting to rely on point solutions to address the problems. One such problem is caused by carrying forward from the client-server world the assumption that, even though the client machines are widely distributed, they are all under the physical control of the owner of the server. That's not true on the Internet, and it's causing problems. Once again, people are proposing point solutions to fix specific aspects of the problem. Hence, the interest in IPSec, SSL, PKI, and a whole host of other acronyms.

6.0 Assumption and Implications

When we started the research on e-speak, we realized that we had a unique opportunity, a chance to design a distributed system from the ground up, knowing that the Internet was there as a commercial platform. Had we started a year earlier, we might not have realized that business could be conducted on the Internet. After all, through most of its life, the Internet was a means to deliver data, not services. None of the earlier designers could have known what we knew; those that followed would be working against long established patterns.

How to start? We decided to ask what assumptions we should make. From them we could construct our design principles from which the architecture would follow. OK, it's never that clean. In fact, it was an iterative process. As the early work proceeded we would re-examine assumptions or introduce new ones. Fortunately, the process quickly converged to the following assumptions.

6.1 Large Scale

We assumed that we would have to deal with 1,000,000 machines. We now know how ridiculous a number it is, ridiculously small, that is. However, thinking about a million machines focuses your attention. We realized that we could not have a centralized anything, not a database, not a control point, not an authenticator. Any such a thing would eventually become a bottleneck.

A very large number of machines also means that you have to give up the idea of keeping data consistent. If you try, you'll spend all your time waiting for changes to propagate through the systems, and you won't get any useful work done. In e-speak, any piece of data you get, even from the local machine, may well be out of date. Most of the time what you get is good enough. If you need current data, go get it. It turns out that if you assume your data is consistent, and it's not, you've got big trouble. However, if you assume your data is inconsistent, it's much easier to deal with out of date information.

6.2 Dynamic

With a million machines, someone is always tripping over a power cord. New services are appearing; old ones are going away. E-speak is built to work in a dynamic environment like the Internet. Every part of the system, from naming to invocation and replies, has special properties for dealing with this world.

Services need to be written differently. In a static world with predetermined configurations, a program can be written assuming that what it needs can be found, and once found, used for as long as necessary. This approach won't work on the Internet. E-speak encourages programming for a dynamic world. If a service isn't found, try again later or compose what you need out of other services. If a service you've been using disappears, find a provider of an equivalent service.

6.3 Heterogeneous

The world is heterogeneous and getting more so, not less as Microsoft and Intel would have you believe. For example, a Saab has 50MB of software, a network, and 12 operating systems. Your electric shaver has 8KB of software and an operating system. You don't even want to think about what goes on inside a rice cooker! The heterogeneity is not just in hardware platform or operating system, though. It is also in functionality. E-speak was designed to work on devices ranging from supercomputers to pagers or even light switches.

6.4 Hostile

The Internet environment is hostile. There are bad people out there; there are careless people out there. Security is difficult to do, so difficult that system designers often leave it for last. After all, if the thing isn't going to work at all, there's no reason to do all the hard work of adding security mechanisms. Unfortunately, if you try to add security later, it is very difficult to block all the ways around the mechanisms.

E-speak started with a way to control access to resources. In fact, the first two machines we connected each ran a script called "malicious", attempting to get the other machine to do something it shouldn't. There are data structures in the code today that appeared in the first prototype. Security is built into the very fiber and being of e-speak.

6.5 Many Domains of Control

There are different fiefdoms. The fact that your security is based on classifications, such as secret and top secret, and mine is based on compartmentalization should not prevent us from cooperating. You should be able to manage your system any way you like; I should be able to do the same with mine.

E-speak addresses this problem by never looking inside another machine. If you ask me in the right language, and permission is granted, I'll do it. I don't care what operating system you're using, or whether you're even running e-speak. You might be a squirrel with an abacus for all that I know. All I care about is that you've asked me to do something that I've agreed to do on your behalf.

7.0 Design Principles

These assumptions led us to a number of design principles. First of all, the three of us weren't going to be able to write a million lines of code to do what we wanted, so we needed to keep things simple. On the other hand, we couldn't forget the problem we wanted to solve. Our solution was to develop a small set of abstractions and use them over and over again. Remarkably, the abstractions that we developed early in the project survived the test of time.

E-speak was designed to deal with technological advances. In order to deal with a dynamic environment like the Internet, e-speak has to be seamless, flexible, and to allow for dynamic evo-

lution, able to deal with current and future requirements. That word, *future*, is very important. There is no need to wait for the next release of the software, no need to shut down the system and restart, even for a kind of thing invented after the system started running.

E-speak is designed to be scalable, manageable, and secure. Designed to operate on the Internet or Intranet, e-speak avoids bottlenecks such as centralized databases and a reliance on data consistency. Its mechanisms provide the hooks necessary to manage and secure the system without forcing everything to be done in a specific way.

E-speak has no special cases. Any special case needs very strong justification, and none passed the test. Each is an architectural nightmare, a development nightmare, a maintenance nightmare. Every time a special case was proposed we would lock ourselves in a conference room until we figured out how to do without it. Doing the work in a research environment gave us the luxury of taking this time, but its benefits contributed to the speed with which the product was produced.

E-speak has a completely uniform access model. E-speak makes no distinction between a service, such as a billing service, and a resource, such as a transaction to that billing service. Even though people normally think of a service as being active and a resource passive, e-speak's mechanisms are the same for both.

E-speak deals only with service control information, not service semantics. If you say "open a file", and permission is granted, your file gets opened. If you say, "open a butterfly", and permission is granted, the butterfly opens its wings. The important thing is that e-speak does the exact same thing in each case. E-speak makes no attempt to understand the semantics of your service. That's why it can deal with a new kind of thing; e-speak doesn't even try to understand an old kind of thing.

E-speak mediates and virtualizes every request. In a dynamic world, things change rapidly. Getting policy information and using it for any length of time can be dangerous. Furthermore, services can come and go at any time. E-speak needs to be able to deal with such occurrences without breaking the application. Mediation lets e-speak keep management informed, and virtualization let's e-speak hide the changes from the application.

E-speak is built on industry standards wherever possible. There is no need to reinvent what the industry has adopted. E-speak leverages transports such as TCP/IP and HTTP, business logic such as XML, and standard ways of encrypting and authenticating.

8.0 E-speak in Perspective

Think about your computer. There's a lot of stuff inside of it, most of which you don't want to know about. One thing that's not inside your computer is anything called a file. There's a disk with tracks and blocks and bytes, but nothing called a file. Yet, you can talk to the operating system about a file. The operating system uses some of that stuff inside your computer to present the *abstraction* of a file.

There is nothing on the Internet called a file. There's a link which refers to a URL. A URL has some components - a protocol, a location, maybe a port number, and a path. Eventually, this all gets converted to a set of bytes in a block in a track on a disk in a computer, but there's no such thing as a file on the Internet. As shown in Figure 6, E-speak lets you present the *abstraction* of a file on the Internet, just the way the operating system on your computer presents the abstraction of a file.

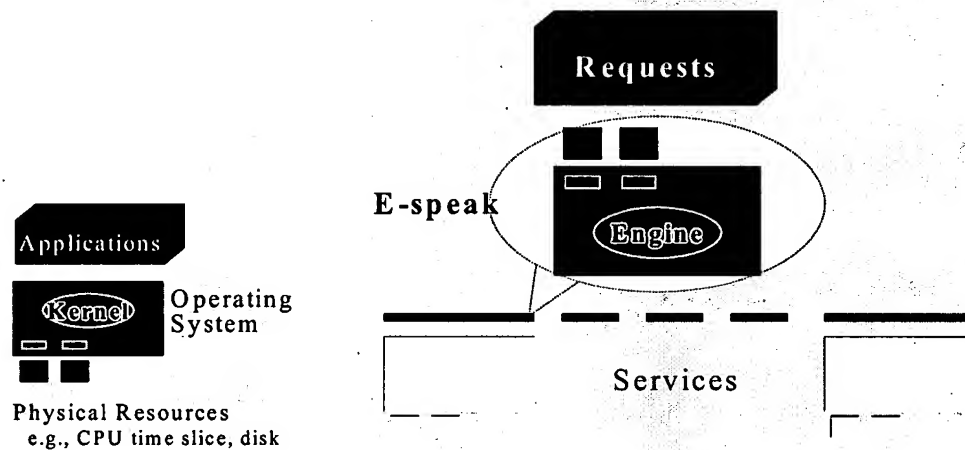


Figure 6. E-speak on the Internet compared to an operating system on a computer.

Think about it. What does your operating system provide? In one view, an operating system does just three things, namely it

1. presents a programming model based on a resource abstraction,
2. provides means to control access to those resources, and
3. talks to the hardware.

E-speak does the first two. In that sense, e-speak is sort of like an operating system for the Internet. It's not really an operating system, but you can think of it as one. If you do, your job of producing an e-service is much easier. Program to the e-speak abstractions as shown in Figure 7, just the way you program to your operating system abstractions today. Someone, perhaps HP, perhaps partners, perhaps competitors, provides the basic set of infrastructure services needed to run an e-service. The e-speak stack takes care of hiding the complexity of the Internet, allowing you to get to market faster with less effort.

9.0 Technology

E-speak is a single, coherent architecture that deals with the four key problems of distributed computing, namely, naming, discovery, security, and management. Here I'll briefly describe how e-speak addresses each of these problems.

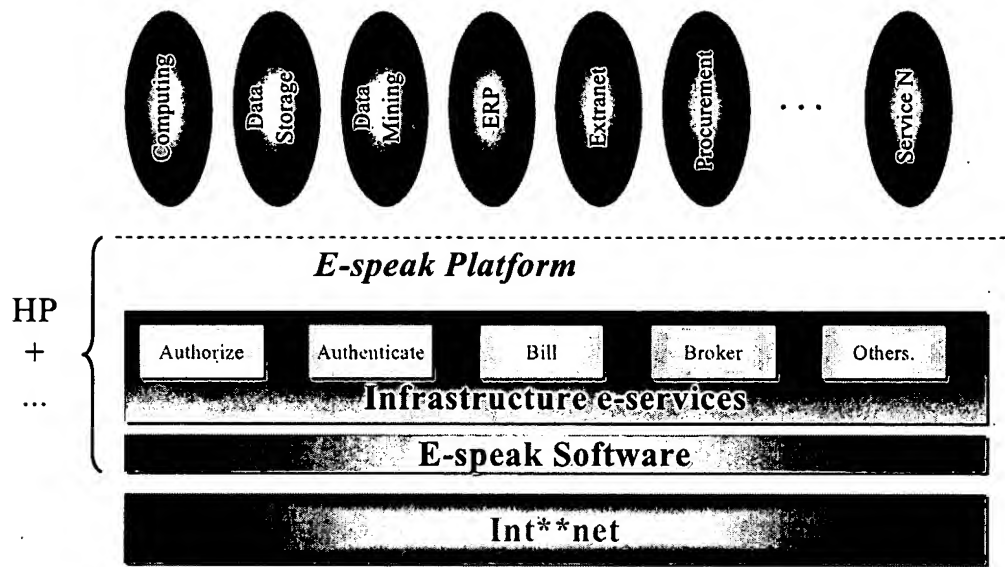


Figure 7. Viewing e-speak as sort of an operating system for the Internet.

9.1 Naming

Most distributed systems give the problem of naming short shrift. Most simply use URLs or something similar. As simple as a Universal Resource Locator sounds, it is not sufficient in a dynamic environment like the Internet. First of all, you have to know the name to use the service. Secondly, the dynamic nature of the Internet means that the service you're requesting may have been renamed, may not be available, or even still in business.

E-speak addresses these problems by mapping each name to an algorithm for finding the kind of service you need. An e-speak name can be bound to a single service, a list of services, a search request for a service, or any combination. Thus, if the service you used yesterday isn't available today, your name binding can be used by the system to find an equivalent provider, all without your application crashing or needing special code to deal with the situation. This example is one way that e-speak lowers the barriers to service creation.

9.2 Discovery

If names aren't used to find services, what is? E-speak encourages you to think of what job you want done instead of how to do the job. Naming a specific service falls into the category of telling how to do the job; finding a suitable provider is telling the system what job you want done. E-speak provides some very powerful mechanisms for discovering service providers.

We all know what's wrong with many search engines; you either get no hits or 40,000. Why? Because there's no context for the search. You're looking for GATES, but are they gates for your garden, gates in a circuit, or are you a lawyer for the US Justice Department? Without a context the search engine must return all matches.

The normal way to provide this context is through a vocabulary. Many systems provide such frameworks, but if it's not complete enough for you to describe your service, you're out of luck until you can get your changes accepted. E-speak allows anyone to define a vocabulary at any time. This vocabulary is an e-speak resource like any other and can be discovered if it is advertised in another vocabulary. (We provide a base vocabulary that everyone understands to get things started.)

E-speak also provides a means to discover services on other machines, an *advertising service*. The advertising service is also used to form *communities*, groups of cooperating e-speak systems. In general, you can find any service offered by any e-speak machine in one of the communities you belong to.

9.3 Security

One reason we have so many problems with security on our systems is that the usage model is wrong; these systems were designed for an environment that didn't include the Internet. Not so for e-speak. We built e-speak for a world where each person wants to share part of a machine with one person and a different part with someone else. We also designed e-speak with the knowledge that bad or careless people would have access to systems belonging to others. Unlike many other systems, security is built into e-speak from the very beginning.

E-speak access control is based on two important concepts, fine granularity and dynamic roles. Fine granularity means that every e-speak service, every e-speak resource can be controlled independently of any others. In fact, individual operations on individual resources can be separately controlled. Dynamic roles recognize the fact that who I am is often less important than what job I'm doing. Sometimes I'm the engineer; sometimes I'm the Daddy. If I change jobs, my access rights as an engineer should go to my replacement. The right to discipline my child should not.

An important part of e-speak's security infrastructure is the ability to have e-speak messages cross corporate firewalls in a secure way. This feature frees the application designer from needing to worry about such issues. Because e-speak separates this process into separate parts, the application part and the security policy part, services are easier to write, and are more likely to work in a changing environment.

9.4 Manageability

E-speak mediates every request. There is a cost in latency, but a tremendous gain in manageability. Since the e-speak engine sees each request, we can rely on it to generate the appropriate management events. There is no need to build this piece into every application, verify that the implementation is correct, and make sure that everything gets updated for every new version.

E-speak provides the tools necessary to convert these low-level events into those appropriate for high-level management systems such as HP's OpenView®. Equally important is the fact that e-speak events are carried as e-speak messages. Since e-speak messages can cross firewalls, so can management events. This feature opens up the possibility of outsourcing system management. It is even possible to have different contractors manage different aspects of a single system. E-speak

security ensures that no information leaks to unauthorized parties and that control of each part of the system is only possible by those given the relevant access rights.

10.0 Summary

E-speak is an environment that provides a number of advantages for service providers and their users. Figure 8 shows how the pieces fit together. The central part is the e-speak engine and the intermachine communication. HP is currently supporting the three platforms shown and provides low level libraries for Java, C, and Python programmers. There is also a high level library for Java programmers and a set of XML schemas that each provide abstractions suitable for creating e-services. The e-speak service bus and infrastructure services are key to making it easier to bring the next layer of applications and services to market.

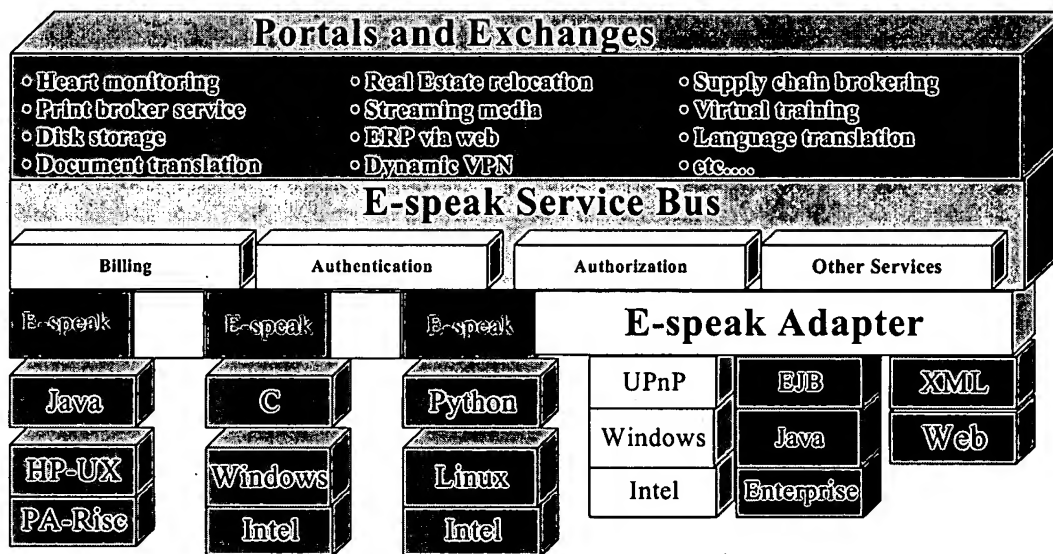


Figure 8. E-speak as a service development platform.

The e-speak adapter was originally developed so that e-speak services could be provided and used solely through the exchange of XML documents. In fact, by translating HTML into XML, we make it possible to provide services to clients who have nothing but a standard browser, one without so much as an e-speak plug-in.

Once we have the idea of an adapter, it's not hard to envision including complementary solutions into the e-speak world. Prototype code has been written that allows any Jini enabled device or Enterprise Java Bean to be an e-speak service without any modification. The effect is that the Jini device or bean can be discovered, managed, and secured using the full power of e-speak. They also benefit from the language independence of e-speak and its ability to work on the Internet, including across firewalls.

In summary, what is e-speak?

E-speak is an operating environment for the Internet that reduces the barriers to creating e-services.

How much does it reduce the barriers? Well, our first customer built the first version of its e-service in 3 months in mid 1999, in spite of our changing interfaces between the Beta 1.0 and Beta 2.0 releases. Our next customer took only 2 weeks in early 2000. Even more impressive was the company that called to say that they downloaded the open source material, studied it for a couple of weeks, and produced a working version of their service in only 2 days.

That's not good enough. The stated goal of e-speak is to reduce the time from idea to market to 2 *hours*. How is that possible? A partner is working on a tool that will allow a business planner to describe the business's logic in a UML (Unified Modeling Language) diagram, something being taught in the business schools today. If every node in that diagram is an e-speak service, the job is done. Talk about reducing the barriers to producing e-services!

11.0 Contributors

E-speak is the work of many hands. This section acknowledges the contributors who have had a significant impact. Since no work succeeds solely on its technical merit, I'll point out the significant management contributions as well.

11.1 Technical

Work on e-speak started at HP Labs in late 1995. Bill Rozas and I put together the first prototype. I coded the earliest version of the e-speak engine in Perl, and Bill produced an e-speak aware file system that included mirroring and code to allow us to move desktops from one machine to another. The latter did not make it into the product, but was instrumental in helping people understand our vision.

Arindam Banerji joined the project in mid 1996, just at the point where we were ready to rearchitect the system. The three of us disappeared into a conference room for a couple of months and produced the architecture that survived largely unchanged through the Beta 2.2 release. Throughout this time, Rajiv Gupta not only managed our department but also made substantive technical contributions. The e-speak e-services vision is largely his.

About the time we started implementing the new architecture, we added Chia Chao and Sandeep Kumar to the team. Chia and Sandeep largely developed the means of matching lookup requests with resource descriptions. Sandeep also produced an in-memory repository, one that survives today. Sven Graupner joined next, first as a student, then as a regular employee. He was the first one to use the new architecture to get two machines talking to each other, in spite of a rather sketchy inter-machine specification.

After HP decided to productize e-speak, several changes were made. Sandeep managed to get a persistent repository working with an Oracle database, while Chia took over improving the engine. Tom Rokicki joined the team and completely reworked the messaging layer. Mike Saboff did the same for the name resolution code. Tom also helped design and implemented the quota

system. Wooyoung Kim took over developing the advertising service and recognized that it could be used to form communities. David Stephenson architected and implemented much of the e-speak management system. Mike Wray and Nigel Edwards produced the architecture for providing end-to-end security that first appeared in the Beta 3.0 release, with Mike doing much of the implementation. Naiem Dahti put together the picture of the e-economy in Section 1.0 and introduced the term “e-economy” to me.

11.2 Management

It is likely that e-speak would have amounted to little more than some conference talks or papers in dusty journals had it not been for Joel Birnbaum, the Director of HP Labs at that time. His support led to the beginning of the research, “How’d you like to work on this for 6 months?”, to bringing executives from HP and a number of companies to see the demonstration we put together, to helping us get funding.

Rajiv Gupta is the kind of manager who will tell you, “That’s the craziest idea I ever heard; let’s try it.”, an important attitude in the early days of the research. Once a decision was made to go to market, he put together an outstanding team, starting with Greg Kleiman as marketing manager and the first person Rajiv brought into the newly formed Open Services (now E-speak) Operation. Rajiv has managed a group that has grown from 6 people in late 1998 to over 200 by mid-2000. All the while, he has continued to push on the people he reports to, often in an environment where software was treated as only an adjunct to selling hardware. It is largely through Rajiv’s efforts that e-speak was released as open source, and he has continually fought to ensure that E-speak Operation is measured on the pervasiveness of e-speak, not revenue.

Arindam Banerji, a relatively new Ph.D. with no previous management experience has led the engine development team. Starting from the unenviable position of having researchers as developers, he put together all the pieces needed for successful product development. In spite of the aggressive, some would say unrealistic targets set for him, his team has yet to miss a deadline. Going from a prototype that almost worked in January of 1999 to the Beta 2.2 release in December, 1999, is a testament to AB’s ability to organize, motivate, cajole, threaten, do whatever it takes to make the targets. He managed to do this in an environment where start-ups were always recruiting his best people. It takes organizational skills beyond the ordinary to absorb the loss of key people without slipping the schedule. The fact that he has done all this with no previous experience managing a project of this scale makes his accomplishments all the more remarkable.

Web E-Speak: Facilitating Web-Based E-Services

Wooyoung Kim, Sven Graupner, Akhil Sahai, Dmitry Lenkov,
Chetan Chudasama, Samuel Whedbee, Yuhua Luo,
Bharati Desai, Howard Mullings, and Pui Wong
Hewlett-Packard

E-Speak, Hewlett-Packard's e-services initiative, is an open, distributed platform that lets e-services dynamically and securely advertise, discover, and interoperate with each other. Web E-Speak, the gateway to E-Speak on the Web, facilitates engineering Web-based e-services by taking into account their requirements for dynamic ad-hoc discovery, secure interaction, and global accessibility.

The Internet's expansion has brought a plethora of e-services to customers' fingertips. E-services are services rendered over a network (the Internet or an intranet), including applications, computing resources, business processes, and information. Until now, these e-services have been static and monolithic and have required human intervention. Soon intelligent e-services will be able to discover other e-services and securely interact with them over the Internet.

E-Speak is Hewlett-Packard's open software platform that will bring this potential to reality. It's a distributed infrastructure on which e-services advertise themselves, discover other services and compose them, and customers discover services and invoke them. (See the "Brief History" sidebar for more background.) This article is a snapshot of the development effort on E-Speak and Web E-Speak. We give an overview of the E-Speak technology and its services and present the Web E-Speak architecture and its functionality.

E-Speak background

E-Speak addresses three important challenges for intelligent e-services. First, it lets heterogeneous e-services communicate with each other and broker services across networks. We call a service E-Speak-enabled when it implements the E-Speak e-services interface¹ and is advertised in an

E-Speak ecosystem, which has e-services and clients connected over a web of E-Speak engines. Because we built E-Speak on Java (<http://java.sun.com>) and the Extensible Markup Language (<http://www.w3c.org/XML>), any E-Speak-enabled e-services can interact with any other regardless of their physical location, platform, system management policy, development environment, or device capabilities. Second, we designed E-Speak from the ground up with security in mind. It provides secure firewall traversal and mediates all accesses to services. Lastly, E-Speak has a service framework that facilitates flexible advertisement and dynamic ad-hoc discovery. The platform and transport independence along with the strong security lets disparate entities on the Internet interact with each other in a secure, manageable way. In addition, flexible abstractions for advertisement and discovery enable service providers to dynamically discover and assemble e-services on the Internet.

E-Speak lies between clients and services and mediates message traffic. Message mediation is crucial for seamless interoperation between clients and services. It frees services from implementing all the communication protocols that a client might use. For example, E-Speak lets both a client implementing the hypertext transfer protocol (HTTP) and one implementing the wireless application protocol (WAP) access a service supporting only transmission control protocol (TCP). E-Speak also abstracts different network deployments between clients and services to allow dynamic n-tier deployment. For example, E-Speak delivers messages to services behind firewalls without disrupting the firewall deployment. Clients or ser-

Brief History

Joel Birnbaum conceived the vision of E-Speak as *pervasive computing* in the early 1980s. In the late 90s, the idea of pervasive computing evolved into the *world-of-services* vision where everything is an e-service, and e-services dynamically discover other e-services and compose them to provide service. The vision led Hewlett-Packard to create E-Speak and make it the cornerstone technology of its e-services initiative. The original vision of Birnbaum's pervasive computing has also percolated into other projects such as the HP Cooltown project (<http://cooltown.hp.com/>), the HP Chai project (<http://chai.hp.com/>), and the MIT Oxygen project (<http://www.oxygen.lcs.mit.edu/>).

vices can use secure HTTP (HTTP/S) for secure point-to-point interaction with E-Speak. E-Speak's session layer security (SLS)¹ provides them with even stronger end-to-end security. By separating out security and communication concerns from applications and e-services, E-Speak offers users cost-effective solutions for integrating heterogeneous legacy applications and aggregating back-office services to bring them onto the Internet.

Web E-Speak is the gateway to an E-Speak ecosystem from the Internet. Web E-Speak defines an XML document content model for e-services to access essential E-Speak services including advertisement, dynamic discovery, and user authentication. Accessing E-Speak services can be as simple as writing an XML document and sending it to Web E-Speak. Service access through document interchange makes clients and services programming-language independent because they communicate using XML documents.

Another important service of Web E-Speak is message routing between clients and services through E-Speak engines. XML's interoperability and simplicity encourage many companies and organizations to develop XML-based document interchange frameworks. In these frameworks, clients (businesses) interact with services (other businesses) by exchanging documents that comply with a published content model. Web E-Speak securely and manageably delivers those documents. In addition, Web E-Speak provides session management and message persistency.

E-Speak's computation model

E-Speak builds its abstractions and system functionality on a single, first-class entity named *resource*.¹ A resource is a uniform representation of an entity registered with an E-Speak ecosystem. In response to a registration request, E-Speak creates a resource that abstracts the entity and associates a message box with it. Once created, a resource resides in a hosting E-Speak engine.

E-Speak doesn't distinguish active services such as name servers, printing services, and car sales' services from passive resources such as files, data entries, and user-profile information. Rather, it serves them uniformly by managing only their representations. For convenience, users can distinguish active services from passive resources.

For example, suppose a user creates a file service and registers it with an E-Speak ecosystem. The corresponding file resource within the system can be a list of path information, file size, and access control policies. The hosting E-Speak engine doesn't

access the file directly but routes and deposits into the message box requests to the file service. The file service listens to the message box for incoming requests and accesses the file on request.

An entity (either an active service or passive resource) is registered with an E-Speak ecosystem with *descriptions* and a *specification*. Each description describes how the service is advertised to users whereas the specification contains access information such as interfaces and access control policies. A description is a set of name-value pairs. A service specification is composed of interfaces describing how clients interact with the service, security policies that prescribe who has what access rights, and a filter constraint that specifies those who may discover the service. On request, descriptions may be advertised to other E-Speak engines but specifications may not. The resource representation's dichotomy allows a flexible yet secure service discovery framework.²

Generally, real-life entities are advertised differently depending on their role or functionality in a context. A wholesale online bookstore might advertise itself as a book seller to retail bookstores and individual customers and as a distributor to publishers. A fax-phone might have two descriptions—one for the fax and the other for the phone. To support e-services with multiple roles or functionalities, E-Speak lets e-services have multiple descriptions and provides clients with a powerful query mechanism in which queries can refer to multiple descriptions.

Services with different descriptions can share the same set of attribute names (that is, the same name space). Moreover, the attribute-based description and lookup leads to a potential name collision problem. For example, suppose one seller advertises a 21-inch monitor and another advertises a 21-inch television. When a user tries to find a television using a query "size=21," she might unexpectedly get the 21-inch monitor as well. To facilitate the name space sharing while avoiding the name collision problem, E-Speak introduces the abstraction *vocabulary*.

Vocabulary defines a set of valid attributes and their types to create a distinct name space. Descriptions must be specified in a specific vocabulary. Attributes in a query are qualified with a vocabulary reference to resolve the name ambiguity. E-Speak engines will validate any descriptions or query constraints against specified vocabularies. In a sense, a vocabulary in E-Speak is to a set of descriptions in the vocabulary what a type in programming languages is to a set of val-

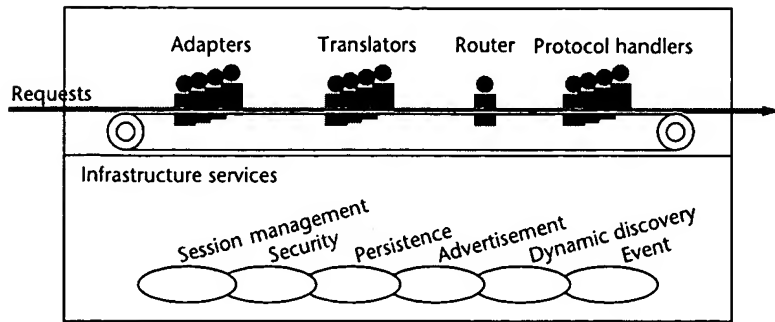


Figure 1. A document interchange engine with a pipeline architecture. The engine contains four pipeline components. The adapters receive inbound documents and translate them to intermediary representations. The translators perform schema translation to resolve potential schema incompatibility. The router delivers a document using an appropriate protocol handler. The protocol handlers implement different transport protocols.

ues in the type. Thus, vocabularies naturally partition the search space of descriptions. This lets vocabularies evolve over time independently of other vocabularies.

Vocabulary itself is a resource and thus is registered and discovered. Because attribute names in a query must be qualified with a vocabulary reference, those who don't know about a vocabulary in which a service is described might not find the service. Thus, service providers can protect their services from being discovered by unwanted intruders by creating their own vocabulary and making it visible only to preferred clients.

If a client requests it, E-Speak virtualizes names that identify services. With name virtualization, neither service providers nor clients must reveal their identity to interact with each other. The hosting engine keeps the mapping from virtual to actual references. Combined with dynamic discovery, name virtualization can provide for dynamic fail-over, runtime upgrades, and service relocation without disrupting clients. Furthermore, we can use it with service replication to implement transparent load balancing.

E-Speak engines mediate all requests to services using the Simple Public Key Infrastructure (SPKI)^{3,4} based attribute certificates. User authentication establishes necessary user credentials. Different access rights to a service can be granted depending on authenticated attribute certificates. If a requested service is behind a firewall, E-Speak enables firewall traversal to route requests to the destination service.

The rudimentary service access in E-Speak is by asynchronous messaging. On top of it, E-Speak libraries build user-friendly interaction models such as the network object model and XML document interchange model. The mediated yet uniform access is the fundamental design principle that lets the E-Speak ecosystem accommodate any type of resource and service.

Web E-Speak document interchange engine

The Internet's laissez-faire nature makes it more daunting for a document interchange engine to achieve seamless interoperability. A client application might use a transport that a service doesn't support. An application might encode messages using a format that a server can't decipher. A client and a service might implement different content models. And a client might generate a document conforming to a service's outdated content model.

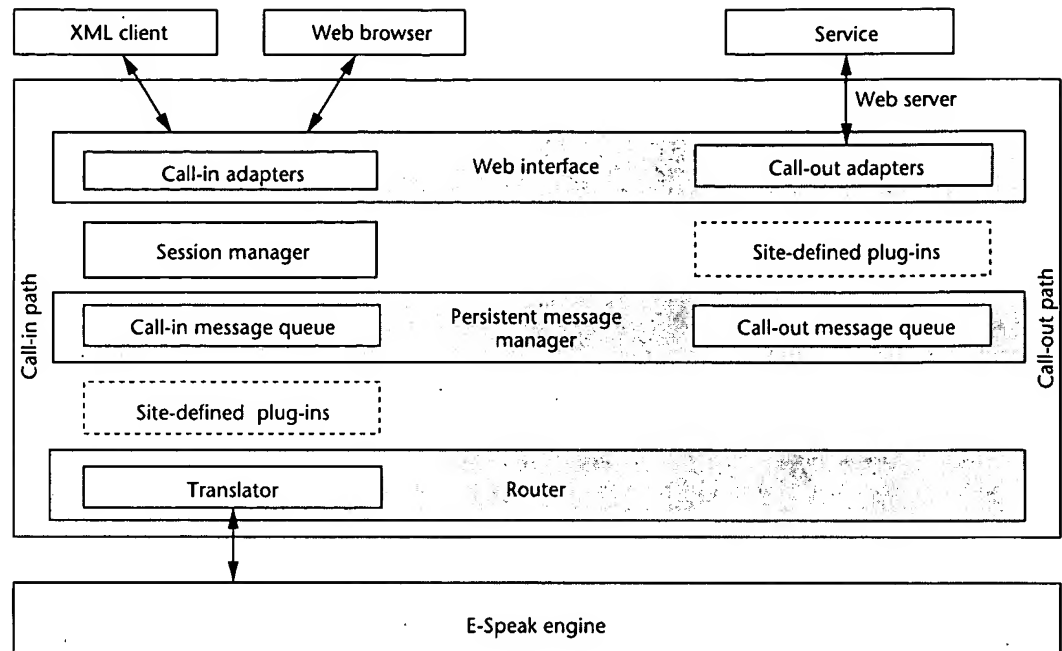
Thus, any document interchange engine needs to address three important issues to be successful in making disparate clients and services interoperate across the Internet seamlessly. First, it should abstract different transport protocols between clients and services. Second, it should translate incoming messages into a uniform intermediary representation to simplify implementation and facilitate message routing. Third, the engine should provide schema translation support to resolve the potential schema incompatibility.

In addition, a document interchange engine might provide some infrastructure services as value-added services, such as reliable message delivery, user authentication and session management, and encryption-based data protection. (Figure 1 illustrates such a document interchange engine.) The engine might even provide dynamic advertisement and discovery mechanisms and broker clients and services.

Overview

Web E-Speak has adopted a pluggable pipeline architecture (similar to one in Figure 1) to quickly adapt to varying requirements in a fast-changing environment. An enterprise or market maker can customize Web E-Speak with site-specific business logic, either home-brewed or from a third party, and reconfigure it dynamically as requirements change. It can also remove unused modules from the

Figure 2. The Web E-Speak architecture. Web E-Speak consists of two parallel pipelines: call-in and call-out paths. The call-in path processes incoming messages, and the call-out path delivers outgoing messages. The two are almost the same except that the call-in path must manage sessions and schema translation.



pipeline and add new modules without disrupting the service. (The current Web E-Speak release implements a statically configured pipeline. A new configuration becomes effective with system restart.)

We implemented Web E-Speak as a Java servlet behind a Web server and made it speak XML over HTTP/S. Messages arrive at one side of the pipeline as an HTTP request and flow to the other end. While a request is in the pipeline, each module looks into it and performs the assigned operation (see Figure 2). Furthermore, the Web E-Speak pipeline is thread-safe. It processes multiple requests concurrently on separate threads that the servlet container in the Web server allocates. We can support larger scale deployments with high scalability or availability by replicating Web E-Speak instances on different server machines.

The Web E-Speak pipeline consists of six modules:

- **Web server.** The Web server provides HTTP/S and Java servlet support. Web E-Speak can use any Web server as long as it supports the Java Servlet API 2.0. We've tested Web E-Speak with Apache 1.3 and JServ 1.1 on Windows NT 4.0, Linux 6.0, and HP-UX 11.
- **Web interface.** The Web interface contains call-in and call-out adapters and translates incoming messages to the intermediary representations and outgoing messages to the client-specific rep-

resentations. Web E-Speak uses the Document Object Model (DOM)⁵ as the uniform intermediary representation. The call-out adapters can have different transport handlers other than HTTP or HTTP/S.

- **Session manager.** The session manager handles sessions for clients and services using the E-Speak account management service. Only authorized users may use Web E-Speak; any attempt to access Web E-Speak is denied if a request doesn't contain a valid session token.
- **Persistent message manager.** The persistent message manager stores requests and replies in a persistent message queue so that services and clients can pick them up when they reconnect to Web E-Speak.
- **Site-defined plug-ins.** The plug-ins are placeholders for modules that may be plugged in the pipeline. Plug-ins provide the primary interface for customizing the Web E-Speak pipeline.
- **Router.** The router resolves destination addresses and delivers messages to their target through E-Speak engines.

Web interface

Clients and services exchange messages in various representations (such as XML, HTML, and

WML). The Web interface module translates these messages to DOM trees⁵ and vice versa. The call-in adapters take incoming messages and generate a vector of DOM trees. Also, they return responses in the same representation as their corresponding request. Clients sending URL-encoded **FORM POST** requests (such as Web browsers) will get HTML responses. Those using XML or multipurpose Internet mail extensions (MIME) messages will receive a response in XML or MIME, respectively. Currently, the Web E-Speak pipeline implements three call-in adapters:

- **HTML/browser adapter.** The browser adapter provides human clients with a simple browser interface that allows HTML/FORM-based access to Web E-Speak. Browsers send requests as a URL-encoded message or a set of HTTP request parameters. The browser adapter translates the requests to DOM trees and returns the replies as HTML documents. Alternatively, it may elect to return XML to Extensible Stylesheet Language (XSL) enabled browsers that can render layouts from XSL stylesheet documents. In that case, Web E-Speak content providers have to provide stylesheet documents separately.
- **XML adapter.** Web E-Speak uses the XML adapter to process requests received as a concatenation of XML documents in a multipart MIME message. The XML adapter converts the sequence of XML documents to a vector of DOM trees. The first XML document must be an E-Speak message header. The rest are considered payload and interpreted only when instructed by the sender. Otherwise, Web E-Speak blindly routes them to target services.
- **MIME adapter.** The MIME adapter lets a sequence of XML documents be sent to Web E-Speak in a single message. It also allows marshalling binary data in a message.

Requests to an external service are passed to a call-out adapter as a vector of DOM trees. The call-out adapter generates XML documents from the vector and delivers them to target services. Depending on the target services, messages are sent as a concatenation of XML documents or as a MIME-encoded document. The XML adapter concatenates the XML documents and sends the result as an HTTP POST request. The MIME adapter passes binary data with XML documents. The call-out adapters use a built-in HTTP handler.

We can plug in other protocol handlers to extend the functionality of the call-out adapters. For example, we can add a simple mail transfer protocol (SMTP) handler to support services receiving requests as email messages.

Session management

Web E-Speak implements an encryption-based secure session management. Only authenticated users may access Web E-Speak. For each request to Web E-Speak (except for an unsuccessful login operation), the session manager returns a new encrypted session token. Any request to Web E-Speak saving a login operation must include the token returned in the response to the latest request. Because session tokens are encrypted (via the Java cryptography extension⁶) and are passed over an HTTP/S connection, malicious clients can't impersonate other clients by guessing session tokens. The session manager sits below the Web interface module and performs session validation for each request by decrypting its token and testing it for expiration. It discards all requests that don't contain a valid token. Login requests are encrypted and passed over an HTTP/S connection to the account manager. On the return path, the session manager intercepts a response message and inserts a fresh token into it, except for a response to an unsuccessful login.

A session token contains all information necessary for session management such as user identification and expiration information. We designed the session token to yield for stateless and lightweight session management. Web E-Speak need not maintain any session state.

The stateless session management allows simple session transfer between a cluster of Web E-Speak engines that share the encryption key. Any requests can be transparently redirected to less loaded engines, improving response time, scalability, and throughput. In contrast, traditional state-based session management (by maintaining session tables) requires that requests in the same session always be routed to the same Web E-Speak engine. Otherwise, costly coherency protocols need to be used to synchronize session tables of different Web E-Speak engines.

Persistent messaging

We can categorize messaging modes in Web E-Speak along three orthogonal axes: synchrony, persistency, and delivery. A sender waits for a delivery receipt when it sends a synchronous message. By contrast, Web E-Speak immediately returns an acknowledgment to a sender when it

receives an asynchronous message. Notice that receiving an acknowledgment doesn't tell that the message has been delivered. Asynchronous messages are used when communication between services is decoupled in time.

Target services of asynchronous messages may not be reachable by the time Web E-Speak attempts delivery. Some senders might come back later and pick up replies (for example, mobile users). Thus, we need to store asynchronous messages in Web E-Speak until their delivery. Web E-Speak uses a persistent message queue to keep these messages. On the other hand, Web E-Speak delivers transient messages with best effort (using a retry mechanism). Transient messages may be lost and not be delivered, but E-Speak doesn't warrant lost-message notification.

Finally, Web E-Speak delivers messages either actively (push) or passively (pull). It always accepts messages from senders and stores them when it can't deliver them immediately. In *push* mode, Web E-Speak initiates delivery to receivers and performs retries if requested. In *pull* mode, Web E-Speak holds messages until receivers connect to it and pick them up.

Eight combinations of messaging modes are possible. The current Web E-Speak release supports the two most frequently used combinations in the real world: synchronous-transient-push (STP) and asynchronous-persistent-pull (APP). The former is primarily for direct, real-time service interaction and has the smallest latency and overhead. The latter is for decoupled, email-like communication where services often have intermittent connection to Web E-Speak.

Messages to Web E-Speak contain delivery information in their header such as synchrony, persistency, push versus pull modes, expiration time, and the number of retries. Web E-Speak stores APP messages in the call-in queue to reduce response time and the probability of message loss. Retrieval by services removes APP messages from the queue. Undelivered APP messages with an expiration time are discarded after the time arrives. Those without expiration time are kept until the default hold period expires. On the other hand, STP messages are stored in the call-out queue after unsuccessful delivery attempt. They remain in the queue until a retry succeeds or all retries fail and the time-out occurs. Web E-Speak returns failure notification in such cases. Replies that Web E-Speak can't deliver immediately are stored in the call-in queue and share the same delivery semantics with APP messages.

Router

The router delivers messages to E-Speak infrastructure services as well as to external e-services. Routing information is contained in the header of a message that conforms to the Web E-Speak message header schema.¹ A message header may have an optional `<content>` element that contains a description of the message's content and/or processing instructions for the message such as the encryption or compression method used. Information the other modules use such as session token, synchrony, and persistency is specified in `<context>` and `<messageInfo>` elements (see Figure 3).

Web E-Speak expects XML requests to be encoded in multipart MIME format when sent over HTTP. The first MIME section should contain an E-Speak message header, the second a request. Binary data may follow in subsequent MIME sections. The router examines the `<to>` element in the header to determine the destination. If the message destination is one of E-Speak's infrastructure services, the request must conform to the Web E-Speak XML document content model and supply an appropriate operation and the corresponding arguments. Web E-Speak will parse the request to validate its conformance and to extract necessary arguments. Requests delivered to external services aren't parsed and are passed on unchanged. This distinguishes Web E-Speak from simple object access protocol (SOAP) messaging⁷ where an optional `Header` and a mandatory `Body` element are enclosed in a mandatory `Envelope` element, forcing an entire SOAP message to be parsed.

In general, a set of document type definitions or schemas that define a service's document content model evolves over time leaving inconsistent requests from clients who don't have access to the up-to-date content model. Upon request, the router applies an optional translation to such requests to resolve the inconsistency before it delivers them. The schema translator in the router applies schema transformation to a request conforming to the target service's old content model and generates an XML document conforming to the current content model.

E-Speak Infrastructure services

E-Speak provides a range of infrastructure services for collaboration of e-services. Here, we describe the infrastructure services that Web E-Speak makes available on the Web.

```

<?xml version="1.0" ?>
<header xmlns="http://www.e-speak.net/Schemas/E-Speak.header.xsd">
  <communication>
    <to encoding="esurl">es://www.web-
speak.com/WebAccess/RegisterService</to>
    <from encoding="url">http://www.mymy.org/~yluo</from>
    <context><sessionToken>sid348098024</sessionToken></context>
    <messageInfo>
      <messageId>mid2832908989</messageId>
      <replyToId>mid1295898999</replyToId>
      <sent>2000-03-21T11:33:01+02:00</sent>
      <MessageProperties synchronous=true persistent=false />
    </messageInfo>
  </communication>
</header>

```

Figure 3. Example message header for a service registration.

Advertisement with service registration

As we discussed earlier, a service provider registers a service with a specification and descriptions. For service registration Web E-Speak defines a document content model compatible with the E-Speak service registration. In addition, the Web E-Speak content model defines the `<locator>` element to specify the service's actual URL. Figure 4 (next page) shows an XML request that registers a person who wants to sell his old car and his old compiler book.

Dynamic service lookup

Users discover services by constructing queries with the attributes of the services and looking up the E-Speak engine for matching services. A query contains a constraint, zero or more preferences, and an arbitration policy. It can have one or more vocabulary declarations if attributes used in its constraints or preferences are from multiple vocabularies. A constraint specifies a condition that services of interest must satisfy. The E-Speak engine applies the preferences collectively to order the results. The arbitration policy specifies how many results are returned (see Figure 5, next page).

A vocabulary declaration associates a local vocabulary reference to a vocabulary. Users use local vocabulary references to disambiguate attributes in a query, which may belong to different vocabularies. A vocabulary declaration is specified with the `<vocabulary>` element, which contains a `name` attribute for local reference and a `src` attribute for vocabulary. In Figure 5, `src` attributes have vocabulary names as their value.

A constraint is a predicate over attributes and

is specified using the `<where>` element. Identifiers in the constraint, such as `make` and `title`, refer to attributes. To distinguish attributes from different vocabularies, we qualify attribute names with a local vocabulary reference.

Once the E-Speak engine finds registered entries that match the lookup constraint, it uses preferences to order them. Preferences come in three flavors. Users use `min` and `max` operators to order matches in ascending or descending order according to an expression. In Figure 5, the client specified that she prefers offers with a low asking price for the car. A client prioritizes preferences using the `with` operator. Users can specify multiple `with` preferences. They can also specify multiple `with` and `min/max` preferences together, offering a flexible preference specification mechanism. (Graupner et al.² gives precise semantics.)

The arbitration policy limits the number of matches returned to the client. A cardinality of positive integer n instructs the E-Speak engine to return at most n services. A client can request the engine to return all matches or any match (randomly selected) it finds. We can use the latter to implement transparent load balancing.

Visibility control through filtering

Service providers generally want to control the visibility of their services to clients. A mortgage lender might want clients with good credit history to find mortgage programs with preferred interest rates. A chip design company might want only its chip designers to find high-resolution plotters and printers. Service providers can specify this kind of visibility con-

Figure 4. A registration request. A seller, his car, and his book are described in the Seller, Used-Car, and Used-Book vocabularies, respectively. The service specification is specified in the resourceSpec element, and the resourceDes element contains a vocabulary declaration and a set of attribute definitions.

```
<?xml version="1.0"?>
<resource xmlns="http://www.e-speak.net/Schema/E-Speak.register.xsd">
  <resourceSpec>
    <locator>mailto:hohoho@es-mail.org</locator>
  </resourceSpec>
  <resourceDes name="John Doe">
    <vocabulary> Seller </vocabulary>
    <attr name="nickname"> <value> old man </value> </attr>
    <attr name="state-province"> <value> California </value> </attr>
    ...
  </resourceDes>
  <resourceDes name="myOldCar">
    <vocabulary> Used-Car </vocabulary>
    <attr name="make"> <value> Honda </value> </attr>
    <attr name="model"> <value> Prelude </value> </attr>
    <attr name="asking-price"> <value> 8,700 </value> </attr>
    ...
  </resourceDes>
  <resourceDes name="myOldBook">
    <vocabulary> Used-Book </vocabulary>
    <attr name="title">
      <value> Compiler: Principles, Techniques, and Tools </value>
    </attr>
    <attr name="asking-price"> <value> 10.00 </value> </attr>
    ...
    <attr name="author"> <value> Ravi Sethi </value> </attr>
  </resourceDes>
</resource>
```

Figure 5. A lookup request. A user is trying to find those who advertise both a used Honda Prelude and a secondhand compiler book by Sethi for sale, hoping that she might get the book for free. She prefers offers with a low asking price for the car.

```
<?xml version="1.0"?>
<esquery xmlns="http://www.e-speak.net/Schema/E-Speak.query.xsd">
  <from src="http://www.john-doe.com/" />
  <vocabulary name="car" src="Used-Car" />
  <vocabulary name="book" src="Used-Book" />
  <result> $serviceInfo </result>
  <where>
    <condition> car:make ="Honda" and car:model="Prelude" and
      book:title="Compilers: Principles, Techniques, and Tools" and
      book:author="Ravi Sethi"
    </condition>
  </where>
  <preference>
    <operator> min </operator>
    <expr> car:asking-price </expr>
  </preference>
  <arbitration>
    <cardinality> all </cardinality>
  </arbitration>
</esquery>
```

```
<vocabulary name="addr" src="Address" />
<filter>$user/addr:state='CA' or $user/addr:state='California'</filter>
```

Figure 6. A filter constraint of a service that states only users in California can find the service. A meta variable \$user is used to refer to the profile information of a user requesting discovery. To find the service, a user should have a description specified in the vocabulary Address and the value of the state attribute in the description must be either CA or California.

```
<variable name="x"> query for a specific discount rate </variable>
<attrDecl name="discountprice" required="true">
  <datatypeRef name="float" dynamic="true">
    <default> price </default>
    <value> price * (1 - $x) </value>
  </datatypeRef>
</attrDecl>
```

Figure 7. Specification of a dynamic attribute. When a user specifically asks for the value of discount_price, E-Speak finds the specified discount rate and computes the value. Also, the value is computed when a query referring to discount_price is given. The default value is used if the expression can't be evaluated due to lack of information.

trol with a filter constraint at registration time (see Figure 6). A filter constraint is a predicate over attributes of a service or resource and may also refer to user profile information. A filter constraint is evaluated when a service matches a user's query. Only those services whose filter constraint evaluates to true are included in the result set.

Dynamic attributes

Most of a service's attributes are static. Their values are fixed when the service is registered and remain unchanged until they are explicitly modified. However, some attributes depend on dynamically changing attribute values of other resources. Some even depend on the requesting user's profile information. These dynamic attributes must be computed at query time. For example, a company might want to let customers find products based on their discount price, which depends on the corresponding discount rate that may change at any time. By specifying an expression that computes the discount price from its retail price (see Figure 7), the company cuts the cost of updating discount prices of their products as the discount rate changes.

Vocabulary creation

A vocabulary is an E-Speak managed resource. Being a resource means it should be described in some vocabulary. Consequently, creating a

vocabulary coerces the creation of another vocabulary that requires another vocabulary and so on. To end the seemingly infinite recursion, E-Speak defines the base vocabulary, which isn't described in any other vocabularies. The base vocabulary is registered a priori across all E-Speak engines. Attributes of the base vocabulary include Name, Type, and Version.

Anybody can create and register a vocabulary; creating and registering a vocabulary is no different from registration of any other services. The only difference is that vocabulary creation requires a vocabulary definition, which defines attributes in the vocabulary. Note that different users can create vocabularies with the same definition, causing conflict between the vocabularies. The vocabulary conflict problem may be resolved with vocabulary unification.¹ Figure 8 (next page) gives a vocabulary creation example.

Account management

The account manager maintains user account profiles and authenticates users against their authentication information (see Figure 9, next page). The current E-Speak release implements a Secure Hash Algorithm (SHA), message digest 5 (MD5) based authentication mechanism. One can configure the account manager to use an SPKI-based mechanism if desired. E-Speak represents users as resources. Thus, registering a new user cre-

Figure 8. A vocabulary creation request. The request will register a vocabulary named Used-Car. It uses the base vocabulary to state that the resource registered is a vocabulary name Used-Car. The vocabulary definition is given in the <attrGroup> element where three attributes (make of type string, model of type string, and asking-price of type float) are defined.

```
<?xml version="1.0"?>

<resource xmlns="http://www.e-speak.net/Schema/E-Speak.register.xsd">
  <resourceDes>
    <vocabulary> http://www.e-speak.net/Schema/E-Speak.base.xsd </vocabulary>
    <attr name="Name"> <value> Used-Car </value> </attr>
    <attr name="Type"> <value> Vocabulary </value> </attr>
  </resourceDes>
  <attrGroup name="used-car"
    xmlns="http://www.e-speak.net/Schema/E-Speak.vocab.xsd">
    <attrDecl name="make" required="true">
      <datatypeRef name="string"/>
    </attrDecl>
    <attrDecl name="model" required="true">
      <datatypeRef name="string"/>
    </attrDecl>
    <attrDecl name="asking-price" required="true">
      <datatypeRef name="float">
        <default>0.0</default>
        <minInclusive>0.0</minInclusive>
      </datatypeRef>
    </attrDecl>
    ...
  </attrGroup>
</resource>
```

Figure 9. User registration request. The request is encrypted and sent to Web E-Speak.

```
<?xml version="1.0"?>
<userNamePass xmlns="http://www.e-speak.net/Schema/E-Speak.account.xsd">
  <userName> John Doe </userName>
  <passPhrase> secret-word </passPhrase>
</userNamePass>
```

ates an account resource in an E-Speak ecosystem that can have multiple descriptions and be discovered just like other services.

E-Speak case studies

Several companies have carried out pilot projects using the E-Speak technology. Here, we give a brief overview of some of the projects. (Further details on these case studies are available at <http://www.e-speak.hp.com>.)

- Hitel, a subsidiary of Korea Telecom, is Korea's leading Internet service provider that offers a game portal service between the country's game houses. Hitel has added dynamic player-ranking, scheduling, and notification services using E-Speak.

- SpinCircuit provides access to an online database of design documents that printed circuit assembly designers can use during the design process. E-Speak lets publishers make design documents available for discovery by other designers. Also, it lets designers securely control and compartmentalize the documents they want to share across the Internet.

- Telia, a leading telecommunication concern in the Nordic/Baltic region, has used E-Speak to provide field service engineers with secure access to their company's back-end system. Service engineers securely retrieve information such as customer work assignment, customer status, and parts inventory through WAP phones using the service.

■ Radiolinja, a subsidiary of Elisa Communications (formerly Helsinki Telephone), is Finland's second largest cellular phone company. Radiolinja used E-Speak to create a mobile e-services ecosystem where users and service providers can dynamically register and discover services accessible through Radiolinja's mobile service hubs. Services in the ecosystem include payment, chat, game, advertising, and auctioning services.

The "Document Exchange Frameworks and E-Commerce Protocols" sidebar (next page) gives additional frameworks and protocols that are well suited for an E-Speak implementation.

Conclusion

E-Speak provides a platform for intelligent service interaction in the Internet. Web E-Speak provides the gateway to the E-Speak community on the Internet. E-Speak uses XML to integrate heterogeneous e-services and smart devices.

The E-Speak product release A.0 was released in January 2001 under the GNU general public license, which we based this article on with the exception of the XML schemas used in the examples that are from E-Speak Beta 3.0 release in May 2000. The software, documents, and other related information are available at the open-source Web site <http://www.e-speak.net> or the HP corporate site <http://www.e-speak.hp.com>.

Despite the fact that E-Speak is an open-source project (<http://www.e-speak.net/>), many in industry view it as Hewlett-Packard's proprietary software, in part because E-Speak has been slow to endorse emerging Web standards. We'll support open standards in the Internet—such as the Web services description language (WSDL) universal description, discovery, and integration (UDDI), and XML digital signature—to make E-Speak an unequivocal open-source platform. Recently, use of mobile computing on wireless networks is gaining momentum in e-services research. We're investigating how to securely support standard wireless communication protocols such as WAP to make E-Speak an information hub to facilitate emerging mobile commerce. **MM**

References

1. *E-Speak Architecture Specification*, version Beta 2.2, Hewlett-Packard, Dec. 1999, <http://www.e-speak.net/library/pdfs/E-SpeakArch.pdf>.
2. S. Graupner et al., "E-Speak: An Enabling Infrastructure for Web-based E-services," *Proc. Int'l*

Conf. Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000), CD-ROM, 2000, <http://www.ssgrr.it/en/ssgrr2000/papers/134.pdf>.

3. C. Ellison, *SPKI Requirements*, Network Working Group RFC 2692, Sept. 1999, <ftp://ftp.isi.edu/in-notes/rfc2692.txt>.
4. C. Ellison et al., *SPKI Certificate Theory*, Network Working Group RFC 2693, Sept. 1999, <ftp://ftp.isi.edu/in-notes/rfc2693.txt>.
5. *Document Object Model (DOM) Level 2 Specification*, version 1.0, W3C Candidate Recommendation, World Wide Web Consortium, May 2000, <http://www.w3.org/TR/DOM-Level-2/>.
6. *Java Cryptography Extension (JCE) 1.2.1*, Sun Microsystems, <http://java.sun.com/products/jce/>.
7. D. Box et al., *Simple Object Access Protocol (SOAP)*, version 1.1, W3C Note, World Wide Web Consortium, May 2000, <http://www.w3c.org/TR/SOAP/>.



Wooyoung Kim is a visiting assistance research professor at the University of Illinois at Urbana-Champaign and has been actively participating in the Java API for XML Registries expert group. His

research interests include e-commerce, parallel and distributed computing, and concurrent object-oriented languages. He was a software engineer/scientist with the HP E-Speak operation when he developed the E-Speak advertisement and discovery mechanism and code-signed Web E-Speak. He has a PhD in computer science from the University of Illinois at Urbana-Champaign, and a BS and an MS in computer engineering from Seoul National University, Seoul.



Sven Graupner is a research scientist in the Internet Computing Technology Center at HP Labs, Palo Alto. His current research interests are in information models and large-scale virtual application environments. He joined the E-Speak project when it was at

an early research stage and was later responsible for Web E-Speak's adapters and communication protocols. He has a PhD in computer science from Chemnitz University of Technology, Germany. He is member of the ACM and the German Computer Science Association (GI).

Document Exchange Frameworks and E-Commerce Protocols

XML's simplicity and interoperability has encouraged companies to work together to develop XML-based frameworks and protocols for automatic information exchange and business execution. E-Speak provides a flexible and secure communication infrastructure on which we can implement the following frameworks:

- **Trading-Partner Agreement Markup Language.** IBM's tpaML automates business-to-business e-commerce collaborations using trading partner agreements. It uses an electronic contract in XML to specify how two trading partners will interact at the transport, document exchange, and business protocol layers. By exchanging TPAs and customized B2B integration software from the TPAs, two parties allow the whole business exchange to take place without further human intervention (<http://www.ibm.com/developerworks/xml/tpaml/tpapaper.pdf>).
- **Electronic Business XML initiative.** This initiative established by United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT, <http://www.unece.org/cefact/>) and the Organization for the Advancement of Structured Information Standards (Oasis, <http://www.oasis-open.org>) aims to develop a framework that standardizes exchange processes of XML business specifications across the Internet and facilitates open trade between organizations, particularly between small- and medium-sized enterprises and developing nations (<http://www.ebXML.org/>).
- **BizTalk.** Microsoft's BizTalk is an XML framework for application integration and e-commerce, which consists of a specification for constructing XML documents and XML schemas for message routing. It achieves interoperability between heterogeneous applications by decoupling issues like document validation and message routing from applications (<http://www.biztalk.org/>).
- **Internet Open Trading Protocol.** The Open Trading Protocol consortium is developing IOTP, a framework for handling electronic payment. The framework's goal is dynamic exchange of payment information between buyers and merchants where merchants outsource payment handling, service delivery, and customer support (<http://www.ietf.org/rfc/rfc2801.txt>).
- **Information and Content Exchange.** The ICE specification provides companies with an XML-based common language and architecture that facilitates automatic exchange, supply, and control of their assets and construction of commercial Web applications across industries such as financial services, publishing, and travel (<http://www.icestandard.org/>).
- **XML/EDI.** XML/EDI is defining how to represent traditional X12 EDI business data using XML. It provides a standard framework to exchange different types of data such as invoices, health-care claims, and project status information (<http://www.xmledi-group.org>).
- **RosettaNet.** RosettaNet is a consortium of companies in the information technology supply chain sector. It has created a common framework and language that all partners in the IT supply chain implement to automate business interactions between them. The framework separates partner interface processes from implementation details such as how to model trading partner agreements between businesses and what public-key infrastructure partners will use, as well as from communication details such as message packaging and message sequencing (<http://www.rosettanet.org/>).
- **eCo.** CommerceNet's eCo framework aims to facilitate business-level integration that enables businesses to publish interfaces, discover other businesses' interfaces on the Web, and determine how they can do business with them (<http://www.commerce.net/>).
- **Common Business Library.** Commerce One's xCBL is an interoperable XML document framework for the cross-industry exchange of business documents such as product descriptions, purchase orders, invoices, and shipping schedules. It lets businesses across multiple industries and marketplaces conduct document-based transactions on the Web (<http://www.xcbl.org/>).
- **Open Financial Exchange.** OFX is a specification for the electronic exchange of financial data between financial institutions, businesses, and consumers via the Internet. It supports a wide range of financial activities including banking, bill payment, investments, and financial planning (<http://www.ofx.net/>).
- **Open Buying in the Internet.** OBI is a standard by which organizations can conduct B2B purchasing transactions over the Internet. It provides a technical framework to automate the requisitioning, approval, order, confirmation, and payment process for goods between organizations (<http://www.openbuy.org/>).



Akhil Sahai is a senior scientist in the software technology lab at HP Laboratories. He is researching Web services management, solutions, and architectures. He was one of the founding members of the E-Speak team that shaped Hewlett-Packard's e-services technology. He has led research at the Multi-Media Systems group at Kent Ridge Digital Labs (previously ISS) Singapore. He has a PhD in computer science from INRIA-IRISA, France, and an ME in computer science from the Indian Institute of Science, Bangalore. He is an IEEE and ACM member.



Dmitry Lenkov is a chief scientist with Talaris Corporation and has more than 20 years of experience in software design and development. Before joining Talaris, he worked as an architect and development lead for the E-Speak project at Hewlett-Packard,

developing metadata, vocabulary infrastructure, and Web E-Speak. He was a chairperson of the C++ standards committee (1989-1995) and participated in technical committees at ANSI/ISO, Open Group, and European Computer Manufacturers Association. He has an MS in mathematics and a PhD in computer science from Leningrad State University, Russia.



Chetan Chudasama is contracting at Fujitsu Network Communications as an XML expert helping them to integrate XML into their product line. His research interests include developing Internet-based applications and Web services. He has a BS in electronics and telecommunications from the University of Pune, India.



Samuel Whedbee is a senior software engineer at WebMD/Medical Manager R&D. Before that, he worked at Hewlett-Packard on the E-Speak WebAccess team. His technical interests include distributed computing, databases, and digital image processing. He has a BS in computer engineering from the University of Florida.

He has a BS in computer engineering from the University of Florida.

Yuhua Luo is a senior software engineer in the area of B2B integration at BEA systems. She is working on standards-based solutions that connect trading partners, and applications. She joined the E-Speak project at HP in 1999 and worked on the XML engine and Web access of E-Speak. She has an MS in computer science from the University of Pittsburgh, Pennsylvania, and a BS in computer science from the Tsinghua University, Beijing.



Bharati Desai is a software design and development engineer at Hewlett-Packard. She focuses on developing Web services employing HP Bluestone's Total-e-Server, Java server pages, Enterprise JavaBeans, HP SOAP messaging, and universal description, discovery, and integration (UDDI). She has an MS in computer science from Wayne State University, Detroit, Michigan.

Howard Mullings is a project manager at Blue Pumpkin Software. He was responsible for the design and implementation of the Web E-Speak pipeline architecture. He has a BS in computer science from Harvard College.



Pul Wong is a software engineer in the Web Services Operation at Hewlett-Packard. He developed the Web E-Speak component. He has a BS in computer science from Utah State University.

Readers may contact Wooyoung Kim at 1304 W. Springfield Ave., Urbana, IL 60801, email wooyoung@cs.uiuc.edu, and Sven Graupner and Akhil Sahai at HP Laboratories, 1501 Page Mill Rd., Palo Alto, CA 94034, email sven_graupner,akhil_sahai@hp.com.

For further information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.

E-Speak – an Enabling Infrastructure for Web-based E-Services

Sven Graupner, Wooyoung Kim*, Dmitry Lenkov, Akhil Sahai

Abstract—The explosive growth of Internet has unearthed new business opportunities and created plethora of e-services. However, the ever-increasing number of e-services in the Internet makes it difficult to advertise and discover them. Moreover, getting e-services to seamlessly interoperate with one another becomes a challenging task. E-speak is Hewlett-Packard's initiative for web based e-services. It is a distributed computing infrastructure for e-commerce that allows e-services to advertise, discover, and interoperate with each other to offer new services in a dynamic, yet secure manner. It provides an XML based open platform for creating, composing, mediating, managing, and accessing Internet-based e-services. The paper overviews the E-speak architecture and its abstractions, such as security, management, advertising service, and event mechanism. We describe the architecture of Web E-speak with its functionality, which embraces XML to interact with the web. In particular we focus on how E-speak supports creation, deployment, and discovery of web based e-services, and how e-services interoperate using E-speak.

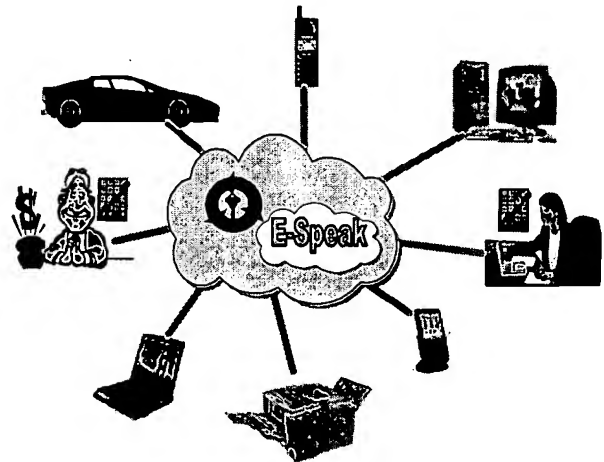
Index Terms—E-speak, e-services, XML, Web

I. INTRODUCTION

The era of standalone e-commerce sites, monolithic and monadic (so called Internet Chapter One), is getting to an end, and the next chapter of the Internet is about to unfold. In Internet Chapter Two, intelligent e-services will interact dynamically, transparently, and securely on the Web to offer a multitude of e-services. E-speak, Hewlett-Packard's e-services initiative, is an open services platform that will bring this potential to reality. It is a distributed platform where e-services advertise themselves and discover and compose others to create new services.

Today's Internet technology falls short in fostering e-businesses and e-services to communicate and cooperate with each other in three areas: support for new business paradigm where e-services securely connect to other e-services, quick deployment/advertisement of new services, and dynamic service discovery and composition. The E-speak architecture addresses each of these challenges. First of all it is designed to let e-services and smart devices communicate and broker services with one another across

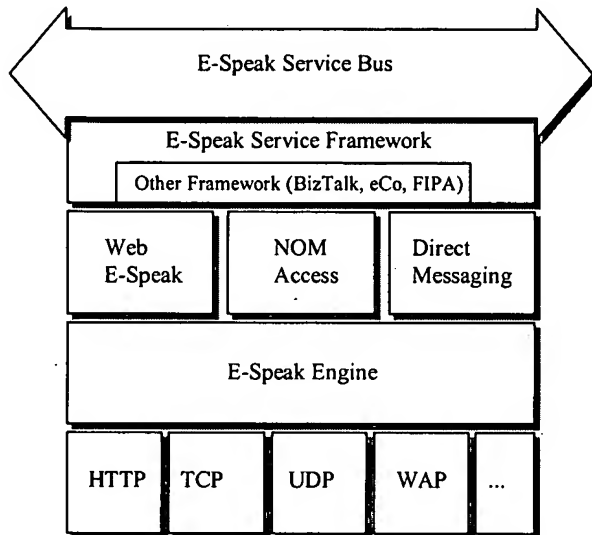
the Internet. It speaks and understands XML, a *de facto* Internet standard, and is developed in Java to leverage its platform independence. Furthermore, it is also designed with security in mind from ground up. For example it mediates all accesses to services and implements secure firewall traversal. The platform and transport independence along with the strong security allows disparate entities in the Internet to interact and provide services for each other in a secure and manageable way. Any E-speak enabled e-services can interact with any other regardless of their physical location, platform, system management policy, development environment, or device capabilities. It provides a flexible resource model for service advertisement and a powerful query mechanism for dynamic service discovery. These abstractions enable e-services in the Internet to be composed and assembled dynamically.



E-speak has a layered architecture where each layer provides a different level of essential abstractions for e-services development. At the bottom is the E-speak engine which provides the basic infrastructure services such as service registration/advertisement, discovery, security, and management. Above the bottom sit a number of programming libraries which support network object and document exchange models. These libraries comprise the E-speak Service Interface (ESI) to the E-speak engine and other e-services. Web E-speak is the one that supports the document exchange model. On top of the libraries is business logic supported by a variety of XML-based

Hewlett-Packard Corporation. E-speak Operation.
10450 Ridgeview Court 49EL-FR. Cupertino, CA 95014, USA.
{sven_graupner|wooyoung_kim|dmitry_lenkov|akhil_sahai}@hp.com
* Presenting author (wooyoung@hpl.hp.com)

frameworks, such as Microsoft's BizTalk Framework [15] and CommerceNet's eCo [4]. A set of protocol and interface specifications that make applications E-speak aware constitute the E-speak service framework. E-speak also defines a service bus that allows e-services to compose themselves dynamically into ad hoc federations to offer new services.



This paper gives an overview of the E-speak, a distributed infrastructure for e-services. In particular we focus on the abstraction in the E-speak engine and the functionality of Web E-speak, a Web gateway to E-speak. Web E-speak embraces XML over HTTP/TCP to interface with Web-based e-services. Two important abstractions, service advertisement/registration and dynamic service discovery, are described in detail in Section III. Then we present an example in Section IV which demonstrates how e-services are composed in E-speak. Finally we summarize related research and draw comparisons with E-speak.

II. OVERVIEW OF E-SPEAK AND WEB E-SPEAK

E-speak builds all abstractions and system functionality on one single first-class entity – *resource*. A *resource*¹ is a uniform representation of an entity created in or registered with the E-speak engine. E-speak does not distinguish active services, such as name servers, printing services, and a car sales service, from passive resources, such as files, data entries, and user profile information. Rather, it treats them uniformly by dealing only with their representations (i.e., *resources*). For their convenience users may distinguish active services from passive resources.

For example, suppose a user creates a file service and registers it with the E-speak engine. The corresponding file

¹ This *resource* is an E-speak term and should not be confused with resources such as files, compute cycles, etc. We will use *resource* to distinguish it from other resources throughout the paper.

resource within the engine is nothing more than a description of the actual file such as name and size, and a specification such as access control policy. The E-speak engine does not access the file directly. Rather, it keeps a mailbox for the *resource* and routes and deposits requests to the file service into the mailbox. The file service defines a handler that listens to the mailbox for incoming requests and accesses the file upon request. In this context, E-speak resembles a runtime system for Actors [1][2][11][12].

Service providers register a service with an E-speak engine by submitting service specification and descriptions. Description is about how the service is presented to users while specification specifies access information such as interfaces and access control policy. The dichotomy of the *resource* representation allows for a flexible yet secure service discovery framework (See Section III). In response to a registration request, the engine creates a *resource* that abstracts the service. The descriptions may be advertised to other E-speak engines. E-speak allows service providers to unregister their services; however, E-speak does not attempt to maintain consistency among multiple engines.

Real life entities have different descriptions depending on their role, functionality, or environment. A fax-phone may have two descriptions, one for fax machine and the other for phone. An online bookstore is a seller to book lovers and a distributor to book publishers. To accommodate the requirement E-speak allows services to have multiple descriptions. A *description* consist of attribute name-value pairs.

Services dynamically enter and leave an E-speak community. Thus E-speak recommends users to discover a service before accessing it. An immediate implication is that every lookup request may return different results. E-speak provides a powerful querying mechanism for users to find most appropriate services (Refer to Section III). A query is specified over descriptions of a service.

The attribute-based description and lookup lead to a potential name collision problem. Suppose a 21-inch monitor as well as a 21-inch TV is registered. When a user tries to find a TV using a query "size=21" she may unexpectedly get the 21-inch monitor as well. To avoid name collision and to facilitate description and query validation, E-speak introduces a powerful abstraction called *vocabulary* and requires descriptions be specified in a specific vocabulary. A vocabulary defines a set of valid attributes and their types in the vocabulary. In other words it defines a name space similar to XML Schema [22][20]. On the other hand a vocabulary defines a set of descriptions in the vocabulary as a type in programming languages defines a set of values in the type. Thus vocabularies naturally partition the search space of descriptions. This allows vocabularies to evolve over time independently of other vocabularies.

Vocabulary is a *resource* managed by the E-speak engine. Since it is a *resource* anybody can create and register a

vocabulary. A vocabulary itself is described in other vocabularies. To end the recursion E-speak defines the base vocabulary with the Type and Name attributes which is unique across all E-speak engines. Since anyone can create vocabularies two identical vocabularies may exist in an E-speak engine. The vocabulary conflict problem may be resolved by vocabulary unification. However, we envision that standard bodies such as RosettaNet [16], CommerceNet [4], and Microsoft BizTalk Framework [15], will create a few globally adopted market-specific vocabularies.

Validity of a description or a query constraint is verified against the vocabulary. Attribute names are qualified with a vocabulary reference to resolve the name ambiguity. Since vocabularies partition the search space of *resources* (i.e., descriptions), a user may share her services only with specific users by creating her own vocabulary and restricting the visibility of the vocabulary.

If requested by a client E-speak virtualizes names that identify services. With name virtualization neither service providers nor clients need to reveal their identity to interact with each other. For each lookup result the engine generates a virtual name and keeps a mapping information. Combined with dynamic discovery, the name virtualization provides for dynamic fail-over, run-time upgrades, and service relocation without disrupting clients. Moreover, it may be used to implement transparent load balancing with service replication.

Accessing services is based on asynchronous messaging. On top of it does E-speak libraries build other user friendly interaction models such as network object model. All requests to services through E-speak engines are mediated. The mediation is enforced using attribute certificates based on Simple Public Key Infrastructure [5][6]. Different access rights to a service are granted depending on what attributes have been authenticated. The mediated yet uniform access is the design principle that allows the E-speak engine to accommodate any type of resources and services.

In summary the E-speak engine provides the following infrastructure services:

- Message routing. Requests and replies are routed through E-speak engines. Reliable messaging may be supported between clients, if necessary.
- Advertisement/registration. Services may be described in multiple vocabularies. The descriptions may be advertised to other E-speak engines. Also services may advertise their descriptions into external advertising depots such as HP E-Services Village via the E-speak advertising service.
- Dynamic discovery. E-speak provides a powerful querying mechanism for clients to use to dynamically discover services from E-speak engines as well as from external advertising depots.
- Security/mediation. E-speak implements an attribute certificate based security mechanism using SPKI to

mediate all accesses to services. If necessary, name translation is used to hide actual identities of service providers and clients.

- Firewall traversal. The E-speak engines implement secure inter-communication across firewalls using Internet standards such as HTTP and Socks V5.
- Persistence. E-speak provides a persistent repository so that *resources* are not lost because of unexpected failures of the engine.
- Events/management. E-speak uses a unique publish-subscribe-distribute model of event mechanism. The E-speak management service uses the event mechanism to provide management functionality for service providers.

Web E-speak, a component of ESI, is a Web interface to the E-speak community. It supports XML over HTTP/TCP to enable seamless access from the Internet. All interactions with Web E-speak are modeled as document exchange. Web E-speak makes the essential functionality of E-speak available to the e-services in the Web. In addition it implements user authentication, session management, and persistent message queue.

Web E-speak defines schemas and a set of XML tags for message routing through E-speak engines. Each message to Web E-speak should contain an XML header that conforms to the Web E-speak message header schema, followed by a specific XML request. The request specifies an operation to be performed as well as arguments to the operation.

Web E-speak consists of a stack of modules. Requests are pushed down the stack until they are delivered to the E-speak engine. The topmost module is the adapter module that generates XML DOM trees from a request. The adapter module is also responsible for transforming FORM POST requests from a web browser to XML documents and XML replies to HTML/WML documents. The generated trees are passed to the session manager where login and logout requests are handled. Only authenticated clients may send messages through Web E-speak. The next module is the persistent message queue manager which handles poll requests. For example it allows mobile clients to have intermittent connection to Web E-speak. Finally the requests are sent to the E-speak engine where they are routed to target services or handled by the engine itself depending on the requested operation. In future releases Web E-speak will support document translation based on XSL/XSLT and dynamic schema lookup.

III. SERVICE ADVERTISEMENT/REGISTRATION AND DYNAMIC SERVICE LOOKUP

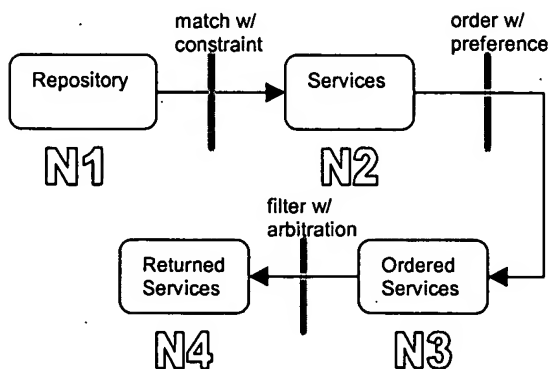
A service is registered with specification and descriptions. A service specification is composed of interfaces, security policies, and a filter constraint. An interface specification describes how clients interact with the service. For example, it may list business logic that the service understands and implements. A security policy prescribes who is granted which access right for a particular service. Service providers may specify those who may discover the

service using the filter constraint. In addition, the actual URL of the service is specified in the `locator` field. A service may have multiple descriptions in different vocabularies. The following request registers an entity known as `Box-A` and as `Container-C`. It is described in vocabulary `box` as well as in vocabulary `container`. The vocabularies are defined as an XML schema and XML namespaces are used to specify the vocabularies.

```
<?xml version="1.0"?>
<resource xmlns="http://www.e-speak.net/Schema/Espeak.register.xsd" >
  <resourceSpec>
    <locator> http://www.parcel.com/box-service </locator>
    <interface> ... </interface>
    <security> ... </security>
    <filter> ... </filter>
  </resourceSpec>
  <element xmlns:box="http://www.parcel.com/box.xsd">
    <box:Name> Box-A </box:Name>
    <box:Length> 20.0 </box:Length>
    <box:Width> 25.5 </box:Width>
    <box:Height> 60.0 </box:Height>
  </element>
  <element xmlns:container="http://www.parcel.com/container.xsd">
    <container:Name> Container-C </container:Name>
    <container:Volume> 30600.0 </container:Volume>
    <container:Content> bubble gum </container:Content>
  </element>
</resource>
```

Users discover services by constructing queries with the attributes of the services and inquiring the E-speak engine of matching services. A query contains a constraint, zero or more preferences, and an arbitration policy. It may have vocabulary declarations if attributes used in the constraint are from multiple vocabularies. The constraint specifies a condition that services of interest must satisfy. The engine applies the preferences collectively to order the results. Arbitration policy specifies how many results are returned.

The following is a pipeline diagram of the lookup process. First the constraint is evaluated to generate $N2$. If preferences are given they are applied to $N2$. If the user specified an arbitration policy it is applied to $N3$ to produce $N4$. If no preference was given, then $N2 = N3$. If no arbitration policy is given, then $N3 = N4$.



A. Vocabulary Declaration

Vocabulary declaration associates a local reference to a vocabulary. Local references are used in constraints and preferences to refer to a vocabulary. A vocabulary declaration is specified with the `vocabulary` element which defines two attributes, `name` and `src`. The `name` attribute specifies a local reference to a vocabulary while `src` specifies how to get the vocabulary. The value of `src` may be the name with which the vocabulary is registered, another query, or an URI which uniquely points to the schema definition of the vocabulary (e.g., XML DTD or XML Schema). Omitting the `name` attribute declares a default vocabulary. Only one default vocabulary is allowed in a query.

```
<vocabulary name="seller" src="PC seller" />
<vocabulary name="buyer" src="http://www.CPU.org/CPUbuyer.xsd"/>
<vocabulary src="http://www.parcel.com/box.xsd"/>
```

B. Constraint

Constraint is a predicate over attributes that may be defined in different vocabularies. To distinguish the attributes, attribute names are qualified with local vocabulary references. For example the following query returns a list of PC manufacturers who sell their products to `ResellerA` and buy CPUs from `CPUMakerB`.

```
<?xml version="1.0"?>
<esquery xmlns="http://www.e-speak.net/Schema/Espeak.query.xsd">
  <vocabulary name="seller" src="PC seller" />
  <vocabulary name="buyer" src="CPU buyer"/>
  <constraint> seller:customer/name = 'ResellerA' and
               buyer:supplier/name = 'CPUMakerB'
</constraint>
</esquery>
```

```
<?xml version="1.0"?>
<esquery xmlns="http://www.e-speak.net/Schema/Espeak.query.xsd">
  <vocabulary src="http://www.vocab-authority.com/pc-vocab.xsd" />
  <constraint> manufacturer='HP' and price < 1500.00 </constraint>
</esquery>
```

Identifiers in the constraint refer to attributes in the vocabulary specified by the `vocabulary` element. An attribute may have an arbitrary XML document as its value. Individual parts in the document are referenced to using XPath [21] expressions.

C. Preference

Once the E-speak engine finds services that match the lookup constraint, it uses preferences to order services. E-speak defines `min`, `max`, and `with` preference operators. The `with` operator takes a condition and a weight expression. The condition is a predicate over attributes. Any numeric expression may be used for latter. A user may specify multiple `with` preferences. For a given *resource*, if the condition of a `with` preference evaluates to `true` the associated weight is added to the total weight of the resource. If evaluated to `false` nothing is added. After all evaluations have been completed the services are sorted in

the descending order of their total weight. For example the following query specifies that a user prefers boxes whose length is less than 5 and whose height is greater than 7, but she prefers taller ones to shorter ones.

```
<?xml version="1.0"?>
<esquery xmlns="http://www.e-speak.net/Schema/Espeak.query.xsd">
  <vocabulary src="box" />
  <constraint> color = 'blue' </constraint>
  <preference operator="with" >
    <condition> Length < 5 </condition>
    <weight> 2 </weight>
  </preference>
  <preference operator="with">
    <condition> Height > 7 </condition>
    <weight> 4*Height </weight>
  </preference>
</esquery>
```

The `min` takes an expression and orders services in the ascending order of the value of the expression. The `max` attribute is the same as `min` except that services are ordered in the descending order. For example suppose a box vocabulary defines three attributes: Length, Width, and Height. If you want boxes with bigger volume, you may specify the preference as follows:

```
<vocabulary src="http://www.parcel.com/box.xsd" />
...
<preference operator="max" >
  <expression> Length * Width * Height </expression>
</preference>
```

If multiple `max/min` preferences are specified in a query the order that the preferences appear in the query becomes significant. The first preference provides the primary order, the second preference is used as a tiebreaker for the result of the first, and so on. The `with` preferences and the `min/max` preferences may also be specified together, but the `with` preferences are given higher priority. Thus the result is sorted according to the `with` preferences first. Then the first `min/max` preference is used to break ties, the second is used to break ties from the first, and so on.

D. Arbitration policy

Arbitration policy limits the number of services that are returned to the client. It is similar to the return cardinality policy in OMG TOS [14]. Three operators are supported: `first`, `all`, and `any`. The `first` operator takes an integer argument so that '`first n`' instructs the E-speak engine to return at most `n` services. The `all` and `any` do not take any arguments and make the engine return all services or any service it finds. The `any` operator may be used to implement transparent load balancing.

E. Visibility Control through Filtering

In general a service provider wants to allow only a certain group of users to discover its services. A mortgage broker may want clients with good credit history to find mortgage programs with the preferred interest rate. A chip design

company may want only its chip design engineers to find very high-resolution plotters and printers. An organization may want workers in the west wing to use printer "nile" and those in the east wing to use printer "ganges" for load distribution.

Visibility control is specified with a filter constraint at registration time. Filter constraint is a predicate over attributes of a service. It may also refer to user profile information. A filter constraint is evaluated when a service matches a user's query. If the result is `true` the service is included in the result set. Otherwise the service is discarded. If the user's profile does not have an attribute that the filter constraint refers to, the constraint evaluates to `false`. In the following XML document fragment, printer "nile" is registered so that only the employees in the west wing may find it. `$user` is a meta variable which is bound to the profile of a user who requests the discovery.

```
<resourceSpec>
  <filter>
    <condition test="$user/loc/loc_in_building='west'" />
  </filter>
  ...
</resourceSpec>
<element xml:printer="http://www.hp.com/e-speak/Printer.xsd">
  <printer:manufacturer> HP </printer:manufacturer>
  <printer:model> LaserJet III </printer:model>
  <printer:name> Nile </printer:name>
</element>
...
```

F. Dynamic Attribute

Most attributes of a service are static; their values are fixed when the service is registered and remain unchanged until they are explicitly modified. However, some attributes are dependent on dynamically changing attribute values of other resources. Some of them depend even on a requesting user's profile information. These *dynamic* attributes need be computed at query time.

For example a company may want to allow customers to find products based on discount price which is dependent on the discount rate at the time of the lookup as the rate may change at any time. By specifying an expression that computes the discount price from its retail price, the company saves updating discount prices of their products as the discount rate changes. Only the update to the discount rate would suffice.

```
<attribute name="discount_price" type="dynamic" default="price">
  <variable name="x"> query for a specific discount rate </variable>
  <expression> price * $x/rate </expression>
</attribute>
```

When a query for PCs is given which refers to `discount_price`, the E-speak engine finds the specified discount rate and computes the discount price. Also the `discount_price` is computed when a user specifically requests for the value. The default value is used if the E-speak engine is not able to evaluate the expression.

IV. EXAMPLE: BOOK BROKER

The Book Broker service is an e-service that provides for comparison book shopping. It discovers online bookstores dynamically and composes them to create its own service. Using the Book Broker example, we demonstrate how E-speak can be used for dynamic composition of existing services in the Internet.

A. Overview of the E-speak Book Broker Service

In this example, the Book Broker service and three online bookstores, Amazon.com, FatBrain.com, and Barnesand-noble.com, are registered as E-speak services. The Book Broker service and the three online bookstores are described in the broker vocabulary and online bookstore vocabulary, respectively. Note that the three online bookstores are not E-speak enabled. We created proxy services to wrap them up and make them E-speak aware. The proxy services are registered with the engine and bridge the Book Broker service with XML over HTTP and the online bookstores with HTML over HTTP.

In the example, clients discover the Book Broker service using the broker vocabulary. They are interested in neither how many online bookstores are currently available in the E-speak engine nor how the Book Broker service finds them. If multiple Book Broker services are registered with the engine, clients may choose one that they prefer.

Upon request from a client, the Book Broker service discovers all the currently available online bookstores and forwards them the book search query in parallel. The online bookstores process the query and return the results to the Book Broker service. The Book Broker service collects all the results and returns a combined list of book offers to the client. Notice that the number and kind of bookstores found may vary over time since online bookstores can connect and disconnect dynamically. The overall architecture of the example is shown in the following figure:

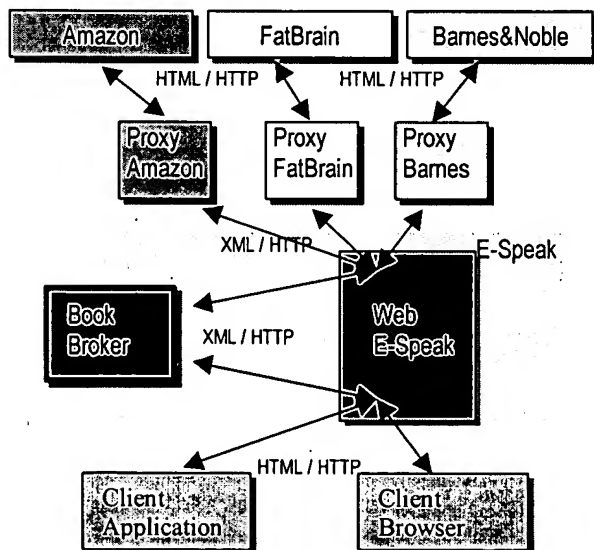


Figure 1: Architecture of the Book Broker service

B. Book Broker Schema and XML Examples

The Book Broker service uses XML documents to represent queries as well as the resulting book lists. The structure of the documents is defined in a simple Book Broker XML schema:

```
<?xml version="1.0"?>
<schema targetNamespace=
  "http://www.e-speak.net/Schema/BookBroker/bookquery.xsd"
  xmlns="http://www.w3c.org/1999/XMLSchema"
  xmlns:offer=
    "http://www.e-speak.net/Schema/BookBroker/bookquery.xsd">
  <element name="bookquery" type="offer:queryType"/>
  <complexType name="queryType">
    <element name="author" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="title" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="subject" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="publisher" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="isbn" type="string"
      minOccurs="0" maxOccurs="1"/>
  </complexType>
</schema>
```

```
<?xml version="1.0"?>
<schema targetNamespace=
  "http://www.e-speak.net/Schema/BookBroker/booklist.xsd"
  xmlns="http://www.w3c.org/1999/XMLSchema"
  xmlns:blist=
    "http://www.e-speak.net/Schema/BookBroker/booklist.xsd">
  <element name="booklist" type="blist:bookListType" maxOccurs="1"/>
  <complexType name="bookListType">
    <element name="bookoffer" type="blist:bookOfferType"
      maxOccurs="unbound"/>
  </complexType>
  <complexType name="bookOfferType">
    <element name="author" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="title" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="publisher" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="year" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="offeredby" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="price" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="shipsin" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="URLtoOrder" type="string"
      minOccurs="0" maxOccurs="1"/>
    <element name="comments" type="string"
      minOccurs="0" maxOccurs="1"/>
  </complexType>
</schema>
```

The example query below finds books with the title "The old man and the sea" by E. Hemingway, which may be

generated from a browser's FORM POST request (See Figure 2) and transformed by Web E-speak.

```
<?xml version="1.0"?>
<bookquery xmlns=
  "http://www.e-speak.net/Schema/BookBroker/bookquery.xsd">
  <author> Hemingway, Ernest </author>
  <title> The Old Man and the Sea </title>
</bookquery>
```

The following is a resulting book list from the Book Broker service.

```
<?xml version="1.0"?>
<booklist xmlns=
  "http://www.e-speak.net/Schema/BookBroker/booklist.xsd">
  <bookoffer>
    <author> Hemingway, Ernest </author>
    <title> The Old Man and the Sea </title>
    <publisher> Bubble Book Publishing </publisher>
    <year> 1996 </year>
    <offeredby> Amazon </offeredby>
    <price> $16.95 </price>
    <shipsin> 3 days </shipsin>
    <URLtoOrder> http://www.amazon.com/.../ </URLtoOrder>
    <comments> paper back </comments>
  </bookoffer>
  <bookoffer>
    <author> Hemingway, Ernest </author>
    <title> The Old Man and the Sea </title>
    <publisher> Sky Publishing Company </publisher>
    <year> 1999 </year>
    <offeredby> Barnes and Noble </offeredby>
    <price> $59.99 </price>
    <shipsin> 1-2 weeks </shipsin>
    <URLtoOrder> http://www.barnes.com/.../ </URLtoOrder>
    <comments> special edition </comments>
  </bookoffer>
</booklist>
```

The Book Broker service returns the resulting book list to clients in XML. Translation from XML to HTML/WML is done by Web E-speak. A sample response screen is shown in Figure 3. Clicking on one of the URLs will take the user to the actual offer at the bookstore's web-site (Refer to the URLtoOrder element).

C. Discussion

Even though the Book Broker service provides simple functionality, it has some notable features.

- Query results are obtained by accessing the live web sites, guaranteeing up-to-date results.
- All the necessary transformation and translation is handled by Web E-speak. Applications may use their own communication protocol and still interact with others without any friction. Document translation using XSL/XSLT between XML documents with compatible XML schemas will be supported in future releases.

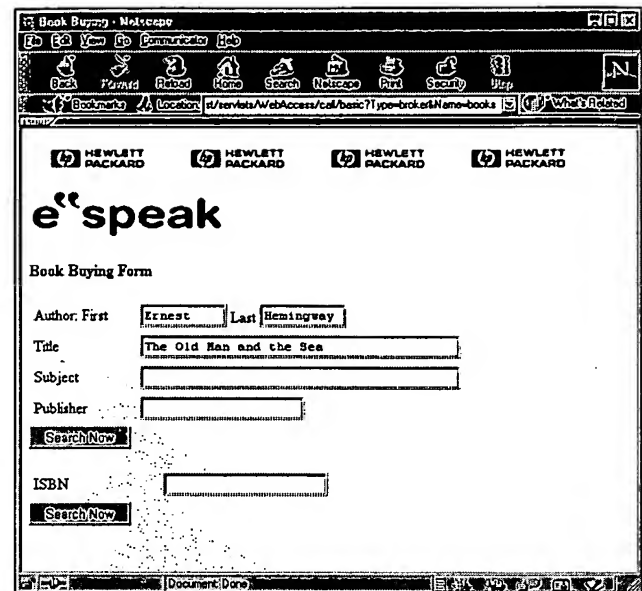


Figure 2: Query screen presented in the browser

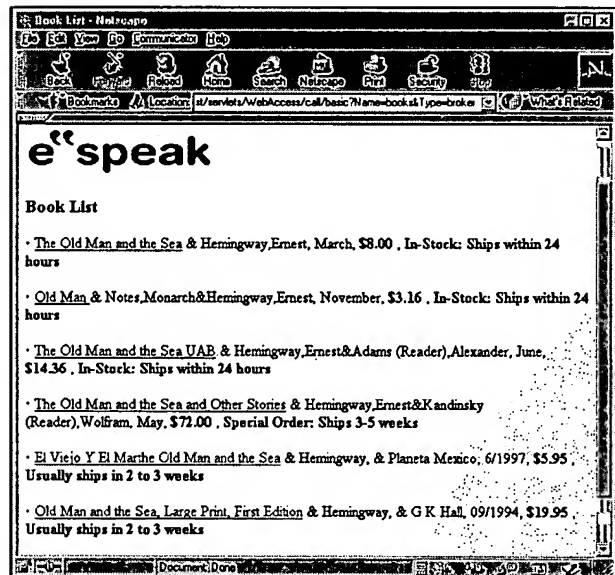


Figure 3: List of book offers returned to the browser

- No prior knowledge about availability and contact location is required to access a service. This information will be obtained on the fly with dynamic discovery. New online bookstores will be instantly discovered as they join the community. Online bookstores leave the community transparently without affecting the Book Broker service.

The proxy implementations are dependent on the contents and the presentations of the online bookstores. The dependency stems from the fact that the contents offered are mixed with their presentation. The dependency results

in the proxies to be re-implemented whenever the online bookstores change their look in the Web. It highlights two challenges that need be addressed in the Web before dynamic service composition is pervasive. First, service providers need to provide open interfaces to contents so that others can have direct real-time access to the contents. Second, contents should be separated from presentation.

The XML technology, an emerging Internet standard, can address these challenges. XML allows service providers to separate content (XML) from its presentation (XSL). Merging with the presentation information can be done using XSLT. We believe e-services in the Internet will eventually separate their contents from presentations and provide XML interfaces to allow direct access to their contents, enabling dynamic service composition.

V. RELATED WORK

E-speak, in particular, Web E-speak, has embraced XML [18][19] to provide interoperability between heterogeneous e-services on the Web. XML is the *de facto* standard for exchanging data over the Internet. It allows representation of data and its structure information in the same text document, obviating the need for other proprietary protocols. Users access E-speak services, such as advertisement and dynamic discovery, by sending XML documents to Web E-speak. Web E-speak defines a generic XML message header to route messages to E-speak enabled e-services.

The OMG Trading Object Service [14] has discovery abstractions similar to those found in E-speak. For example OMG TOS defines service offers that resemble service specification and descriptions in E-speak. However, services must be pre-configured before they are discovered. Also OMG TOS does not have any abstraction corresponding E-speak vocabularies that define name spaces and support a form of security.

Jini technology [3][17] at Sun Microsystems is a set of protocol specifications, which allows services (especially, devices with computing capability) to announce their presence and to find others. It defines protocols for discovery and join, lookup, distributed events, transaction, and leasing. The Jini system requires RMI and Java serialization based communication. Since its discovery protocol is based on multicast, its use is currently limited to local area networks. Moreover, it assumes trusted devices and services, and does not have any security mechanism. Jini supports a primitive template-based lookup with exact match. E-speak provides mediated secure access to services and more versatile querying mechanism.

IBM's Tspaces [10][23] is network middleware which aims to enable communication between applications and devices in a network of heterogeneous computers and operating systems. It is a network communication buffer with database capabilities, which extends Linda's TupleSpace communication model with asynchrony. It is implemented

in Java to leverage its platform independence. T-Spaces supports hierarchical access control on the TupleSpace level. E-speak provides more flexible and secure certificate-based security mechanism.

Microsoft BizTalk Framework [15] is an XML framework for application integration and electronic commerce. It includes a technical specification for constructing XML documents, and schemas and a set of XML tags for message routing. Like E-speak, it assumes that application integration takes place using a loosely coupled, message-passing approach. BizTalk de-couples document validation, data format translation, schema transformation, message routing, and transaction support from applications and assumes that underlying infrastructure provides these supports. By implementing BizTalk Framework's message routing tags, E-speak may be used as an infrastructure of BizTalk Framework. BizTalk Framework is static and supports integration between trading partners who know each other a priori to their interaction. E-speak allows e-services to dynamically discover and interact with other e-services.

HP's Cool Town project [8][13] and IBM's Universal Information Appliance project [7] are two research efforts that connect smart devices or information appliances to information services and let them work together seamlessly. They have specific requirements for communication infrastructure such as dynamic discovery, message routing, local persistence, and event mechanism. These requirements are already supported in E-speak. Thus E-speak may serve as a communication infrastructure for these systems even though E-speak is being developed primarily for interaction between e-services in the Internet.

VI. CONCLUSION

E-speak, an HP's strategic technology for its e-services initiatives, provides a platform for intelligent service interaction in the Internet. Web E-speak serves as a gateway to the E-speak community from the Internet. E-speak uses XML on the wire to integrate heterogeneous e-services and smart devices.

Three basic abstractions presented in this paper enables intelligent service interaction: dynamic discovery, composition, and access mediation. Dynamic discovery allows on-time discovery of services. Service composition integrates existing services to offer new services. Access mediation provides for secure and reliable service access.

E-speak Beta2.2 was released as open source in December 1999. E-speak Beta3.0 is to be released in May 2000 under the GNU GPL license. The Book Broker software that demonstrated how service composition can be done in E-speak has been in the E-speak source code since E-speak Beta 2.2 and will also be available in Beta3.0. The software, documents, and other related information can be found at <http://www.e-speak.net>.

VII. ACKNOWLEDGEMENT

We thank Samuel Whedbee for reviewing the paper. We also thank the other HP ESO Web E-speak team members, Chetan Chudasama, Bharati Desai, Yuhua Luo, Howard Mullings, and Pui Wong, for their devotion to build and test Web E-speak, a component of the E-speak platform.

VIII. REFERENCES

- [1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.
- [2] G. Agha, S. Frølund, W. Kim, R. Panwar, A. Patterson, and D. Sturman. Abstraction and Modularity Mechanisms for Concurrent Computing. *IEEE Parallel and Distributed Technology: Systems and Applications*, 1(2):3-14, May 1993.
- [3] K. Arnold, et. al. *The Jini™ Specification*, Addison-Wesley, June 1999.
- [4] CommerceNet Inc. eCo Architecture for Electronic Commerce Interoperability. Sep 1999. <http://eco.commerce.net/specs/spec.cfm>.
- [5] C. Ellison, SPKI Requirements. September 1999. Network Working Group RFC 2692. <ftp://ftp.isi.edu/in-notes/rfc2692.txt>.
- [6] C. Ellison, et. al., SPKI Certificate Theory. September 1999. Network Working Group RFC 2693. <ftp://ftp.isi.edu/in-notes/rfc2693.txt>.
- [7] K. F. Eustice, T. J. Lehman, A. Morales, M. C. Munson, S. Edlund, and M. Guillen. A universal information appliance. *IBM Systems Journal*. Vol 38, No. 4, IBM Corporation. Oct 1999. <http://www.research.ibm.com/journal/sj/384/eustice.html>.
- [8] Hewlett Packard Corporation. The Cool Town project homepage. <http://cooltown.hp.com/>.
- [9] Hewlett-Packard Corporation. E" Speak Architecture Specification. Version Beta2.2. December 1999. <http://www.e-speak.net/library/pdfs/E-speakArch.pdf>.
- [10] IBM. T-Spaces homepage. <http://www.almaden.ibm.com/cs/TSpaces/index.html>.
- [11] W. Kim and G. Agha. Efficient Support of Location Transparency in Concurrent Object Oriented Programming Languages. In *Proceedings of Supercomputing '95*, 1995.
- [12] W. Kim. *THAL: An Actor System for Efficient and Scalable Concurrent Computing*. PhD Thesis, University of Illinois at Urbana-Champaign, May 1996.
- [13] T. Kindberg, et. al., *People, Places, Things: Web Presence for the Real World*. Hewlett-Packard Laboratories. IAST. 1999. <http://cooltown.hp.com/papers/webpres/WebPresence.htm>.
- [14] OMG. CORBAServices: Common Object Services Specification Chap. 16. Trading Object Service Specification. <ftp://ftp.omg.org/pub/docs/formal/97-12-23.pdf>.
- [15] D. Rogers, *BizTalk Philosophy*. Microsoft Corporation, <http://www.biztalk.org/resources/danroessay.asp>.
- [16] RosettaNet. RosettaNet Implementation Framework (RNIF). Dec 1999. <http://apps.rosettanet.org/library/econcert.nsf/docid/rost-4a76du>.
- [17] Sun Microsystems. Jini Specifications. <http://www.sun.com/jini/specs/>.
- [18] World Wide Web (WWW) Consortium. Extensible Markup Language. <http://www.w3c.org/XML/>.
- [19] World Wide Web (WWW) Consortium. Extensible Markup Language (XML) 1.0: W3C Recommendation. Feb 1998. <http://www.w3.org/TR/1998/REC-xml-19980210>.
- [20] World Wide Web (WWW) Consortium. Name Spaces in XML. W3C Recommendation. Jan 1999. <http://www.w3.org/TR/1999/REC-xml-names-19990114/>.
- [21] World Wide Web (WWW) Consortium. XML Path Language (XPath) Version 1.0: W3C Recommendation. Nov 1999. <http://www.w3c.org/TR/xpath>.
- [22] World Wide Web (WWW) Consortium. XML Schema Part 1: Structures. W3C Working Draft. April 2000. <http://www.w3c.org/TR/xmlschema-1/>.
- [23] P. Wyckoff, S. W. McLaughry, T. J. Lehman and D. A. Ford, "T Spaces", *IBM Systems Journal*. Vol. 37, No. 3, 1998. <http://www.ibm.com/java/education/t-spaces/t-spaces.html>.



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
 United States Patent and Trademark Office
 Address: COMMISSIONER FOR PATENTS
 P.O. Box 1450
 Alexandria, Virginia 22313-1450
 www.uspto.gov

APPLICATION NUMBER	FILING OR 371 (c) DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
09/504,374	02/15/2000	Stephen Corey Wren	

John Henry Muetterties
 7796 S. Datura St.
 Littleton, CO 80120

CONFIRMATION NO. 7814




OC000000013183198

Date Mailed: 07/08/2004

NOTICE REGARDING CHANGE OF POWER OF ATTORNEY

This is in response to the Power of Attorney filed 05/10/2004.

- The Power of Attorney to you in this application has been revoked by the applicant. Future correspondence will be mailed to the new address of record(37 CFR 1.33).


 SONJA M WILLIAMS
 3600 (703) 305-2272

OFFICE COPY

- found in last 3 months
 - fees.
 → withdrawn IDS

HP Web Services Architecture Overview

Kannan Govindarajan, Arindam Banerji

Email: kannang@hpl.hp.com, axb@hpl.hp.com

Abstract:

Web services are different from traditional distributed computing models. Web services architectures provide a framework for creating, and deploying loosely coupled applications. One of the consequences of the loose coupling is that any entity that a web service may interact with may not exist at the point of time the web service is developed. New web services may be created dynamically just as new web pages are added to the web and web services should be able to discover and invoke such services without recompiling or changing any line of code. In this position paper, we outline some of the high-level architectural requirements of a comprehensive framework for web services, we propose a layered approach to architecting web services that allows for pluggability and interoperability. In addition, we distinguish between infrastructure services and application specific frameworks.

Introduction:

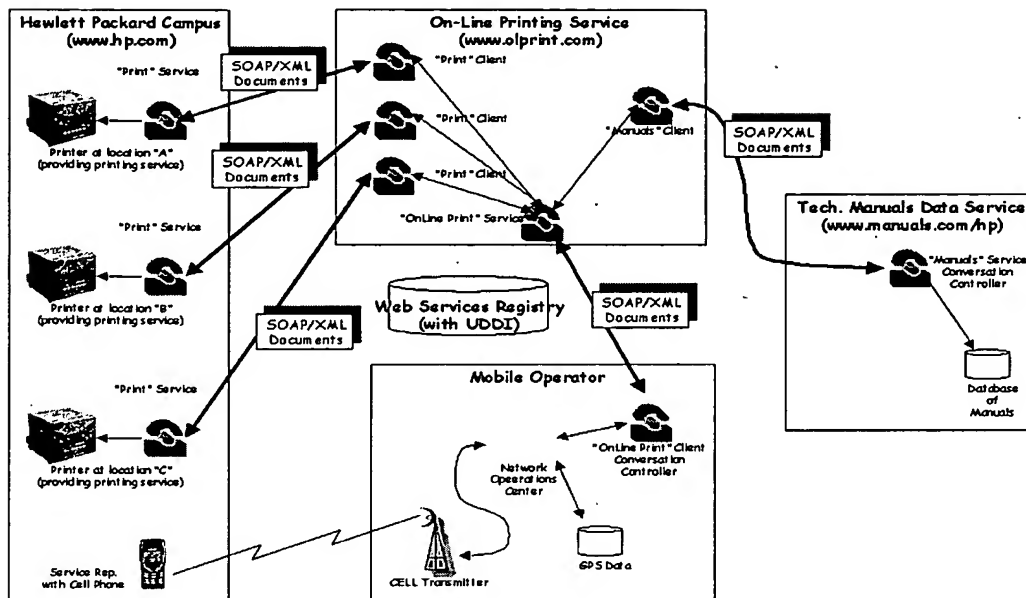
The web service architecture allows businesses to expose business assets as services. Standardizing interactions amongst services has the added advantage that any enterprise can out-source parts of its operation that it does not have expertise in. In addition, since the vision of web services enables web services to dynamically find new web services that it can interact with, enterprises can find new providers for the service relatively quickly. A specific application of this dynamism is in the e-procurement arena. For example, the average sourcing/procurement cycle in enterprises is of the order of 3-4 months. Of this time, about 50% of the time is spent in identifying the appropriate suppliers, about 20% of the time in handling the RFQ (request for quotes) process, and an additional 10% of the time is spent in negotiating the appropriate deal. The ability to dynamically find suppliers can translate to significant time savings, and therefore to lowering of costs. Essentially, the procurement and fulfillment business process are modeled as services, and a hub is the aggregation point for the services. In such an architecture, finding a new supplier is the same as finding the fulfillment service of the supplier at the hub. HP's web services vision enables such a dynamic world by allowing business processes to be modeled as web services, by providing a platform for hosting such web services, by defining the technical conventions that enable the interoperability between web services, and by defining a hub, or aggregation mechanism for web services.

It is our position that fundamentally different component model is required for modeling web services. This is because the assumptions that are made by traditional distributed component models are violated by web services. In addition, we believe that a comprehensive web services platform has at least three related technologies:

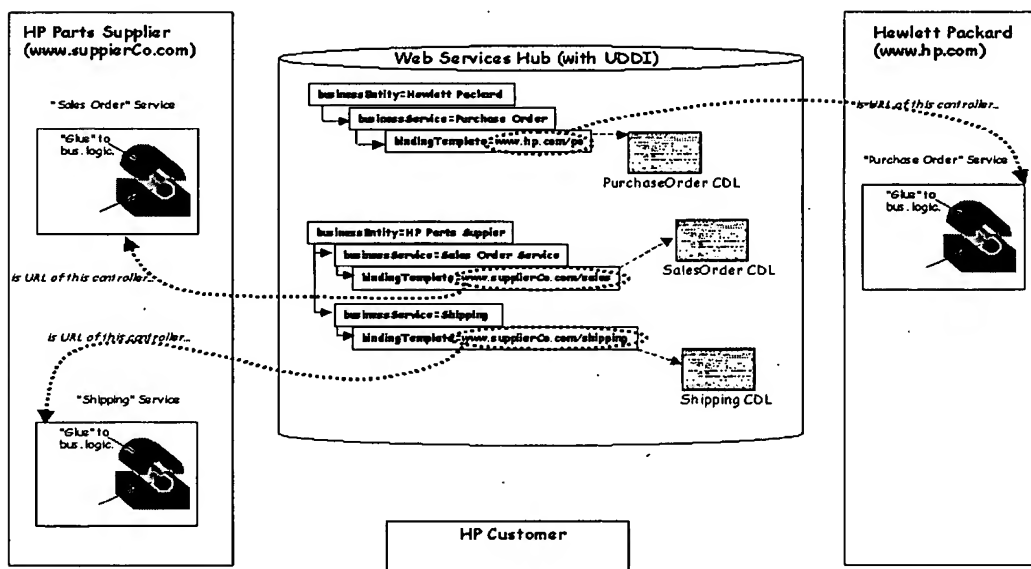
1. The technologies that define the hosting platform that hosts services. Service providers typically will host their services on this hosting platform.
2. The technologies that define the hub that allows services to dynamically discover other services and establish trust in the context of the community. This hub technology potentially is compatible with other hub-like efforts such as UDDI (www.uddi.org).
3. The technologies that define the standard conventions that ensure that services can inter-operate with each other irrespective of their implementations.

The hosting platform provides, among other things, technologies that are required to model existing business asset/process as a web-service, allows clients of web services to invoke services, etc. The hub provides technologies for web services to be described, discovered, etc., and the standard conventions specify the things that have to be standardized so that web services hosted on various web service platforms inter-operate. We at HP have found these three technologies to be extremely relevant in the mobile services and electronic marketplace deployments. We now briefly provide an overview of these scenarios. The mobile service we consider provides a sales representative access to manuals from his cell phone and allows him to print the manuals on a printing service that may be determined by his current location. For example, the architecture of a mobile online printing solution may look as follows:

Components in Architecture



Electronic Marketplace Architecture



The architecture of an electronic marketplace looks as shown above. Note that the two architectures above are quite similar, the MNO in the mobile solution serves as the aggregating and intermediating entity, whereas the hub in the electronic marketplace serves as the aggregation point and may intermediate the access to services hosted on it. We suggest that W3C standardize the technical conventions that enable web services to inter-operate with other web services. The conventions that need to be standardized fall into two categories: infrastructural and domain specific. Infrastructural conventions should be separated into many layers that provide the functionality that essentially allow web service creation, deployment, interaction, and execution. In addition, specific web service infrastructures may also standardize common functionality that is applicable in specific domains. Two domains of particular interest in the web services space are electronic marketplaces, and mobile services. In addition, it is important for W3C to keep all the web services related conventions co-ordinated in order to enable a coherent platform.

Infrastructure for Web Services

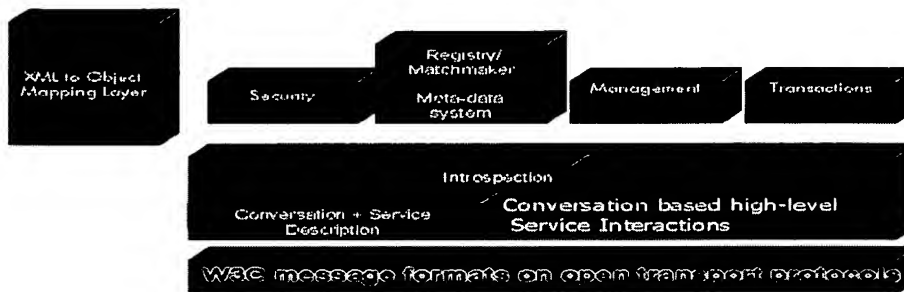
The basic infrastructure for web services serves the same purpose that CORBA and COM serve for traditional distributed computing infrastructures. However, there is an important distinction between the infrastructure for web services as opposed to traditional infrastructures. The inter-operability problem among web services is more challenging than the inter-operability between traditional distributed enterprises. One reason for this is that web services may be separated by firewalls and the semantics of data being communicated between them may not be uniform at the communicating ends. Another important distinction between web services and traditional distributed component models is that the binding between various web services is looser and can occur later than the binding between various components of a traditional distributed application.

Assumptions

Note that one of the key differences between the traditional distributed computing models and web services infrastructure is that the message formats need standardization. This is in addition to standardizing the APIs needed to access the functionality. Standardizing just the APIs as the Java™ distributed computing platform has done leads to a situation where there are many incompatible implementations. For instance, at the time of writing this position paper getting two JMS implementations to communicate was still an open issue. We believe that the message formats should all be characterized as XML messages and W3C guide their definition so that different web service infrastructure implementations can inter-operate. In addition, as mentioned before, web services are fundamentally different from traditional distributed models for the following reasons:

- **Web services are loosely coupled:** Changes to a web service should not require re-installation of software components by the users of the web service.
- **Web services require dynamic binding:** Typically, application designers bind software components to one another at development time. Web services, on the other hand, are likely to be implemented and provided by different service providers. In addition, we must enable easy changes to the services we are using, easy discovery of new services, of new capabilities of existing services, and of new binding or location information of services.
- **Web services communication is based on a Document Exchange Model:** Traditional component frameworks support a network-object model of interaction in which objects of strictly defined types are transferred between components using a request-response interaction pattern. Cross-organizational business interactions do not fit this framework well for two reasons. The interfaces of services may need to be changed in ways that cannot be captured by simple extensions. This precludes the use of object inheritance to support the inter-operability in presence of change. Secondly, interactions can be long lived. Therefore, asynchronous exchange of XML documents is better suited for cross-organizational business transactions.
- **Web services in different enterprises are likely to use different semantics.** The interpretation of the data communicated among enterprises is different for each enterprise. For instance, the address field of a purchase order may have different significance for the parties. If a uniform object model is used, the semantics of data often tends to be similar or homogeneous contributing to tighter coupling.
- **Web services in different enterprises require a distributed model of security.** Security responsibilities are split amongst the enterprises. Each enterprise manages its end of the security infrastructure independently.
- **Web services from different enterprises may be built using heterogeneous technology stacks.** Each enterprise decides on the computing infrastructure independently taking into consideration many factors.
- **Web service interactions must be able to traverse corporate firewalls.** Traditional distributed systems are tuned for applications that are deployed within the enterprise; Web services may be deployed from behind Firewalls. They may need to access other web services across enterprises. For example, in the J2EE™ architecture, the application are written to receive input from outside the enterprise, however, the applications do not often initiate outbound requests to applications in other enterprises. In the realm of web services, accessing applications in other enterprises is the norm rather than the exception.

Web Services Infrastructure Stack



As shown in the picture above, the infrastructure for web services is made up of the following layers:

The communication layer: The communication layer should be transport independent though bindings for common transport protocols such as HTTP, SMTP, etc., should be defined. Essentially, this layer answers the question: How do web services communicate with each other over standard protocols? The messaging layer provides the requisite properties of the messaging such as reliability, security, etc. In order for web services to communicate with each other they have to agree on the technology, standards and protocols for communication. They also need to agree on the syntax and semantics of data they are going to exchange. However, the data that they exchange can be classified into infrastructure parts that the infrastructure uses in order to route the message and application specific parts that the infrastructure does not inspect.

The Service definition layer. Essentially, this layer is concerned with defining how services expose their interfaces as well as the service invocation model. It addresses question such as do the web services expose a RPC like invocation model or do they expose a completely asynchronous document exchange model. The invocation model rests on the messaging model. Since the programming model for web services is more likely to be document exchange based, it is difficult to adopt an object interface definition language for the purpose of defining interfaces of web services. Another aspect to keep in mind when defining interfaces in terms of message exchanges is that the order of message exchanges plays a role in the interface definition. In addition, the security infrastructure has to provide mechanisms that allow service providers to easily determine whether the invoker has the authorization for invoking the service. It should be noted that the service definition layer provides the platform on which the other infrastructure services and functionality is built on. We at HP have invented CDL (Conversation Definition Language) as a means of expressing the interface of web services in terms of the documents that are exchanged with the service.

Introspection: The web service infrastructure has to define mechanisms that allow clients of web services to introspect web services in order to determine how they should interact with them. There are two conversations that allow services to get information about another service they are interested in. These two conversations support the dynamic interaction by allowing a functionality that is similar to the notion of introspection in traditional object systems. The ServicePropertyIntrospection conversation provides general information about the service, the conversation it supports, its provider, and how to access the service. The ServiceConversationIntrospection conversation returns the complete description of the conversations as CDL documents.

The mechanism for binding to services: This allows clients of services to determine the service they want to use in a declarative manner separate from the interface definition of the service. This essentially involves defining the meta-data of services and registry infrastructure for storing and querying the meta-data of services. Matchmaking is the process of putting service providers and service consumers in contact with each other. The matchmaker is where services that want to be dynamically discovered register themselves or their service offerings, and where services that want to find other services send their request for matches. Some of the services advertising themselves through the matchmaker will be simple end-providers, while others may be brokers, auction houses and marketplaces which offer a locale for negotiating with and selecting among many potential providers. The matchmaker is a very simple,

foundational, service on which the rest of the service framework rests, and should be as neutral as possible. It is the web-spider search engine of the web services world. The matchmaker service depends on the meta-data definition system that is provided by the infrastructure. This meta-data system should be flexible and capable of defining the meta-data of a wide variety of services. Some of the requirements of the meta-data definition system are the following. It should be capable of defining the metadata for a wide variety of services. This means that if different vertical industries want to define their metadata for the services in their industry in different ways, the mechanism should allow that. It should allow the metadata to evolve in a gradual manner without changes to the backends or the enterprises that are using old versions of the metadata definition. This allows specific advertisers to differentiate their descriptions of the goods/services they sell with characteristics that enhance their advantage over their competitors. It should be compatible with existing metadata definition mechanisms so that existing web services can be advertised at matchmakers and discovered. It should identify portions of the metadata that are searchable

The security infrastructure: This enables web services to secure various aspects of the service. Traditional PKI infrastructures may not be suited for the web services space. In addition, the security infrastructure will have to provide not only secure transport, but also higher-level access control mechanisms. Web Services or E-Services pose a unique challenge to the design of a distributed security system. At the high level the security system performs the access control, accountability, system integrity and confidentiality.

Unfortunately security systems today cannot do these things well in a distributed environment that spans multiple, independently managed security domains. Some of the specific problems that must be solved are:

- Elimination of man in the middle attacks at boundaries of security domains. Frequently security must be compromised to convert from one security model to another at a border of one or more security perimeters.
- End to End authorization and confidentiality
- Accountability
- The system must not rely upon any central security domain. There are known problems with global name spaces, interoperability of dissimilar security domain infrastructures, the need for trusted third parties at boundaries, etc. The system's objective is to establish an authorization, accountability, and confidentiality agreement between a client and service without depending upon any centralized security domain.
- The system needs to be suitable (scalable?) for a wide range of service value. A service can range from a few cents to potentially millions (billions?) of dollars. While multiple security systems could be invented, tailored to the needs of low or high value services, the challenge is to develop a single security infrastructure that can be readily adapted to either extreme.

Transaction Support: The web services infrastructure should provide support for some notion of transactions so that web services can compose other web services in a transactional manner. Traditional two phase commit protocols may not be appropriate in the context of web services. In addition, transactionality in the web services world may be restricted to providing support for atomicity only. This is in contrast to the consistency, isolation, and durability properties guaranteed by traditional distributed transaction systems. Some of the questions that need to be answered by the support for transactions are:

- Do we allow resource locking?
- What is the model for achieving transactions?
- What protocols do we need to support in order to achieve atomicity?
- Do we make compensation time-bounded?
- Does atomicity apply to conversations or isolated invocations?
- Should we enable composition of different atomicity models?

Management Support: The web services infrastructure should specify how web services are to be managed so that various aspects of the interaction among web services can be managed. There are two interpretations of manageability for services. *Manageability from a management system's perspective* refers to whether a service provides sufficient information (events, measurements, and state) and control points (lifecycle control, configuration control, etc) so that a management system can effectively monitor and control the behavior of the service. There is a second notion of manageability – *manageability from a peer service's perspective*. When two services interact with each other, one of them initiates a request (client) and the other services the request (service). From a client's perspective, a service is manageable if the latter provides sufficient visibility and control over itself and over the transactions or conversations it executes. For example, a service that provides information about the progress of a client's ongoing transactions and

an ability to escalate the speed of transactions in progress (perhaps by paying additional price) is more manageable than a service that does not. Similarly, a service that can be queried about how much quality of service (QoS) it can guarantee and that can provide response time measurements about its transactions is more manageable than a service that does not provide these features. From a service's perspective, a client is manageable if it can provide enough information about service usage back to the service. For instance, a client that can be queried about its perception or quality of experience with the service is more manageable than a client that cannot be. Similarly, a client that provides end-to-end response time measurements of its transactions back to the service is more manageable than a client that does not.

Domain Specific Infrastructure

We now turn our attention to some of the domain specific functionality that the web services infrastructure should standardize. There are two main domains that we believe web services have a major impact. These are: electronic marketplaces, and mobile web services.

Electronic Marketplaces

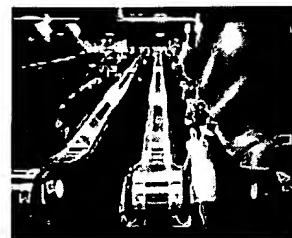
In the specific domain of integrating business applications across enterprises, web services can play a role in standardizing some of the interactions between businesses. The web services infrastructure provides facilities to find web services and interact with web services. However, in the business to business scenario, the negotiation and contract formation functionality has special relevance.

Negotiation: Standardizing the protocols required for various forms of negotiation such as auctioning, two-party negotiation, etc., allows web services to cleanly attach their own specific strategies while still being able to negotiate with a wide variety of parties in order to complete business transactions. The negotiation framework aims to provide infrastructure that allows two or more independent entities to interact with each other over time to reach agreement on the parameters of a contract. It is aimed primarily, though not exclusively, as a means to reach trade agreements. It can be used both by automated entities, and by users via appropriate software tools. Its value to negotiation participants is that it is a prerequisite to provide decision support or automation of the negotiation, and hence make the process more efficient. Furthermore, they can be confident that the basic rules of interaction in any negotiation are standardised, hence reducing the effort to automate many different kinds of business interactions. They are able to negotiate simple contracts, where only price is undetermined, and more complex contracts where many complex parameters depend on each other. Furthermore, the protocols provide the participants with trust guarantees, that no party has access to extra information or is able to forge false information. Its value to negotiation hosts such as auction houses and market makers is that it provides a standard framework that all potential customers can use to interact with them. However, it does not require a specific market mechanism, so allows the host to decide on an appropriate one. It not only provides standard off-the-shelf market mechanisms such as the English auction, but also allows custom mechanisms to be implemented for particular special needs such as the FCC auction for auctioning bandwidth.

Contract Formation and Business Composition: The central idea of the conceptual model for contracts is that the business relationship that motivates interactions that follow is captured explicitly in an electronic contract. An electronic contract is a document formed between the parties that enter into economic interactions. In addition to describing the promises that can be viewed as rights and obligations by each party, the e-contract will describe their supposed respective behavior in enough detail, so that the contract monitoring, arbitration and therefore enforcement is possible. The terms and conditions appearing in the e-contract can be negotiated among the contracting parties prior to service execution. In this way businesses with no pre-existing relationships can bridge the trust gap, be able to strike deals and take them to completion. Business composition is the ability for one business to compose e-services, possibly offered by different providers, to provide value-added services to its customers. Composition of e-services has many similarities with business process (workflow) automation. A web service virtualizes the customer interaction aspects of the business processes implementing a service. In order to specify how services should be composed, we must define the interaction process associated with a service as well as the flow of service and inter-service invocations.



hp web services
platform, an hp
netaction product



hp web services platform: a comparison with hp e-speak

executive summary

HP is one of the pioneers in Web services technology. In the spring of 1995, HP Labs began working on e-Speak, one of the first products focused on Web services. E-Speak was subsequently released to the marketplace, and HP started working with customers to solve complex needs in the area of B2B collaboration.

E-Speak, an HP NetAction product, enables service-to-service interaction by allowing the registration, discovery, and interaction of dynamic Web services. The industry is now focusing on the same capabilities through standardized efforts that include J2EE™ (Java™ 2, Enterprise Edition), XML (Extensible Markup Language), SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery and Integration), and WSDL (Web Services Description Language). With the convergence of industry support on these emerging standards, HP is aggressively working to integrate these standards into its Web Services Platform.

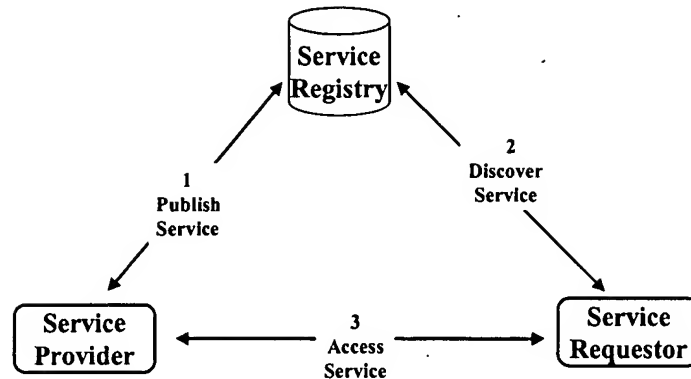
This document discusses e-Speak features and compares them to the features provided in the next generation Web Services Platform. The primary audience for this document is the current e-Speak developers community. Current e-Speak developers who want to migrate to HP Web Services platform may also find this document useful.

introduction

In a typical Web services ecosystem, a Web service provider registers its service with a service registry; using standard vocabularies, a service requestor discovers the required Web service already registered with the service registry; and the service requestor accesses the Web service through one of two mechanisms: Network Object Model (NOM) or Document Exchange Model (DEM). NOM enables legacy systems (such as an Enterprise JavaBeans™) to be accessed as a Web service over the Web. DEM uses XML technology to allow Web services and service requestors to exchange XML documents and thereby to be very loosely coupled.

In e-Speak, the J-ESI (Java e-Speak Service Interface) library allows a developer to use NOM and Gateway (or Webaccess) or to use DEM for registration, discovery, and invocation of Web services. In the HP Web Services Platform, NOM is supported via SOAP XML/RPC (Remote Procedure Call), and DEM is supported through synchronous and asynchronous messaging. The Web service responds to an XML request immediately after processing, and returns an XML response along the same HTTP connection as was used by the incoming request.

The following figure illustrates the conceptual model.



e-speak and hp web services platform features compared

service registry

E-speak developers use the e-Speak service engine (with or without advertising service) or e-Services Village (ESV) to register a Web service. Web services could be dynamically discovered and then invoked by e-Speak clients. In this model, Web services could potentially be deployed in the same operating environment as is used by the Web services registry. Such deployment can, potentially, result in better performance of service invocation.

Developers using the HP Web Services Platform will use a UDDI registry to register and look up Web services. The UDDI registry only stores the URL of the Web service. The service requestor must perform operations on the Web services hosted at the URL returned by the UDDI registry. HP's expertise in building the e-Speak service engine and e-Services Village-based service registry is leveraged in UDDI.

messaging and transport

E-Speak uses its proprietary protocol and messaging infrastructure to communicate with other e-Speak installations (service engines and e-Speak clients). The Gateway usage model allows a non-e-Speak client to interact with an e-Speak service engine using XML over Standard HTTP protocol. The Gateway usage model of accessing the e-Speak Service Engine is very limited, however, in comparison to the J-ESI model that uses its proprietary protocol and messaging infrastructure.

The HP Web Services Platform, which uses SOAP, supports both NOM and DEM models. NOM is supported using the SOAP XML/RPC mechanism, while DEM is used to allow arbitrary XML document exchange in a loosely coupled fashion. To achieve interoperability, two parties must agree on a number of conditions — timeouts, re-try policies, and so forth. This agreement can be embedded into the SOAP header. SOAP, however, does not define how headers are used. Standards groups and vertical industry consortiums, such as RosettaNet, ebXML (Electronic Business Extensible Markup Language) and BizTalk, define such a header, sometimes known as profile. Because of e-Speak, HP's experience is being leveraged in our participation in various standards and their expert groups, such as RosettaNet and JAX (Java API for XML).

vocabularies

E-Speak has a concept of vocabularies for defining Web services. E-Speak vocabularies enable the dynamic discovery of Web services.

The HP Web Services Platform does not use the concept of vocabulary. Instead, it depends on vertical industries and standards bodies to define vocabularies that enable unambiguous interactions during XML document exchange and service lookup based on keywords. In the HP Web Services Platform, UDDI tModels have a role similar to an e-Speak contract. (Contracts are e-Speak entities that contain a set of interfaces that can be discovered or looked up in Service Directories.) A standards body or industry consortium would define a tModel that would have the URL for the interface definition (a WSDL document). Service providers would implement services conforming to these, and the unique tModel key would act as a "signature" for the service.

firewall traversal

E-Speak has built-in firewall traversal capability. An e-Speak client outside the firewall of an enterprise could access a Web service hosted within the firewall without creating any hole on the firewall. This native e-Speak capability, in conjunction with e-Speak security, provides a secure mechanism of traversing a firewall with very fine-grain access control.

The HP Web Services Platform leverages firewall traversal from the proxy listeners in application servers. HP Total-e-Server comes with such a listener.

security

When business applications are exposed to the Internet, security becomes extremely important. Authentication, authorization, and encryption should all be taken into consideration. E-speak provides great infrastructure for fine-grained security. It is a proprietary implementation of SPKI (Simple Public Key Infrastructure). E-speak provides transport level security as well as fine-grained access control (up to method level) security. This security framework is well suited for end-to-end security in NOM; however, it is not well suited for DEM in Web services because the document itself is not secured.

The HP Web Services Platform provides standard XML digital signature-based security. It can use the HTTPS transport protocol to provide confidentiality of the message. The HP Web Services Platform focuses on industry-standard security infrastructure to provide interoperability. Instead of supporting one security mechanism for all programming models and platforms, the HP Web Services Platform supports a combination of various standard security models, including the UDDI, EBXML, and SOAP security models.

wsdl

The e-Speak Services Framework Specification (SFS) provided a framework for creating, deploying, and invoking Web services. Conversation Definition Language (CDL), a part of SFS, is one of the first few formal languages used for the definition and interaction of Web services.

The HP Web Services Platform supports WSDL. IBM, Microsoft, and Ariba originated WSDL as a formal language that would facilitate the definition and interaction of Web services. As the standards evolve, the HP Web Services Platform will support other Web service definition languages as well.

presentation framework

E-Speak does not provide any presentation framework for Web services. An e-Speak developer is left to transform data returned by the back-end Web services for presentation to the browser or any Web client.

The HP Web Services Platform supplies an elegant presentation framework, based on Cocoon2, which facilitates the separation of content, logic, style, and management. It also provides a framework for data mapping. Such data mappings include XML-to-XML, XML-to-relational data, relational data-to-XML, XML-to-objects, and objects-to-XML.

summary

With its next generation product, Web Services Platform, HP has leveraged its expertise in e-Speak and offers support for critical Web services standards such as SOAP, UDDI, and WSDL. The new HP Web Services Platform emphasizes XML and J2EE, the continued leveraging of standards, and a shift from solution implementation to creating an integrated middleware infrastructure that supports HP's service-centric computing strategy. The new software platform will help solve the same business problems that e-Speak was designed to solve, and HP's alignment with industry standards will provide increased interoperability to the users of the HP Web Services Platform.

FOR MORE INFORMATION

For information on e-Speak architecture, features, and functionality, visit the e-Speak Web site at <http://www.e-speak.net>. For information on HP Web Services, visit <http://www.hp.com/go/webservices>.

Java is a U.S. trademark of Sun Microsystems, Inc. Microsoft is a U.S. registered trademark of Microsoft Corporation. All other brand names are trademarks of their respective owners.

Technical information in this document is subject to change without notice. All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

© Copyright Hewlett-Packard Company 2001



Position Papers for the World Wide Web Consortium (W3C) Workshop on Web Services

W3C Web Services Team
Contact: Harumi Kuno
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-73
April 3rd, 2001*

E-mail: hkuno@hpl.hp.com

Web-services,
W3C, E-services,
transactions,
electronic
commerce

This technical report is a collection of position papers that HP submitted to the World Wide Web Consortium (W3C) Workshop on Web Services. The W3C Web Services Workshop represents a community interested in XML-based Web service solutions and standardization of components thereof, including both solution providers and users. The goal of this workshop is to advise the W3C about which further actions (Activity Proposals, Working Groups, etc.) should be taken with regard to Web services.

Introduction

The W3C Web Services Workshop represents a community interested in XML-based Web service solutions and standardization of components thereof, including both solution providers and users. The goal of this workshop is to advise the W3C about which further actions (Activity Proposals, Working Groups, etc.) should be taken with regard to Web services.

In order to present a coordinated position, researchers from HP Labs (both Palo Alto and Bristol) and representatives from both the E-Speak Organization and the E-process Operation collaborated closely (by telephone and email) to produce nine position papers reflecting HP's position on topics ranging from conversational interfaces for web-services to electronic contracts to service management. This technical report is a collection of the position papers that HP submitted to the World Wide Web Consortium (W3C) Workshop on Web Services. (The order in which they appear reflects the infrastructure stack sketched in the "HP Web Services Architecture Overview" paper.)

Table of Contents

CONVERSATION DEFINITIONS: DEFINING INTERFACES OF WEB SERVICES	10
ADVERTISING AND DISCOVERING BUSINESS SERVICES	15
REQUIREMENTS FOR AUTOMATED NEGOTIATION	21
TOWARDS THE ELECTRONIC CONTRACT	25
SECURITY REQUIREMENTS FOR WEB-SERVICES	28
A FRAMEWORK FOR BUSINESS COMPOSITION	34
TRANSACTIONAL CONVERSATIONS	39
A PEER-TO-PEER SERVICE INTERFACE FOR MANAGEABILITY	45

HP Web Services Architecture Overview

Kannan Govindarajan, Arindam Banerji

Email: kannang@hpl.hp.com, axb@hpl.hp.com

1. Abstract:

Web services are different from traditional distributed computing models. Web services architectures provide a framework for creating, and deploying loosely coupled applications. One of the consequences of the loose coupling is that any entity that a web service may interact with may not exist at the point of time the web service is developed. New web services may be created dynamically just as new web pages are added to the web and web services should be able to discover and invoke such services without recompiling or changing any line of code. In this position paper, we outline some of the high-level architectural requirements of a comprehensive framework for web services, we propose a layered approach to architecting web services that allows for pluggability and interoperability. In addition, we distinguish between infrastructure services and application specific frameworks.

2. Introduction:

The web service architecture allows businesses to expose business assets as services. Standardizing interactions amongst services has the added advantage that any enterprise can out-source parts of its operation that it does not have expertise in. In addition, since the vision of web services enables web services to dynamically find new web services that it can interact with, enterprises can find new providers for the service relatively quickly. A specific application of this dynamism is in the e-procurement arena. For example, the average sourcing/procurement cycle in enterprises is of the order of 3-4 months. Of this time, about 50% of the time is spent in identifying the appropriate suppliers, about 20% of the time in handling the RFQ (request for quotes) process, and an additional 10% of the time is spent in negotiating the appropriate deal. The ability to dynamically find suppliers can translate to significant time savings, and therefore to lowering of costs. Essentially, the procurement and fulfillment business process are modeled as services, and a hub is the aggregation point for the services. In such an architecture, finding a new supplier is the same as finding the fulfillment service of the supplier at the hub. HP's web services vision enables such a dynamic world by allowing business processes to be modeled as web services, by providing a platform for hosting such web services, by defining the technical conventions that enable the interoperability between web services, and by defining a hub, or aggregation mechanism for web services.

It is our position that fundamentally different component model is required for modeling web services. This is because the assumptions that are made by traditional distributed component models are violated by web services. In addition, we believe that a comprehensive web services platform has at least three related technologies:

1. The technologies that define the hosting platform that hosts services. Service providers typically will host their services on this hosting platform.
2. The technologies that define the hub that allows services to dynamically discover other services and establish trust in the context of the community. This hub technology potentially is compatible with other hub-like efforts such as UDDI (www.uddi.org).
3. The technologies that define the standard conventions that ensure that services can inter-operate with each other irrespective of their implementations.

The hosting platform provides, among other things, technologies that are required to model existing business asset/process as a web-service, allows clients of web services to invoke services, etc. The hub

provides technologies for web services to be described, discovered, etc., and the standard conventions specify the things that have to be standardized so that web services hosted on various web service platforms inter-operate. We at HP have found these three technologies to be extremely relevant in the mobile services and electronic marketplace deployments. We now briefly provide an overview of these scenarios. The mobile service we consider provides a sales representative access to manuals from his cell phone and allows him to print the manuals on a printing service that may be determined by his current location. For example, an architecture of a mobile online printing solution is shown in Figure 1:

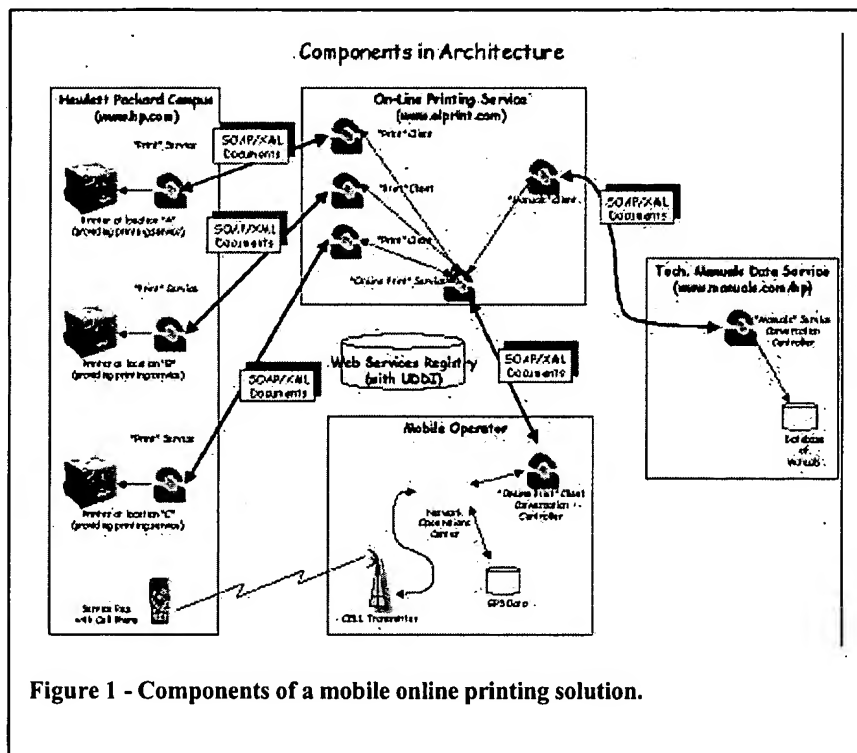


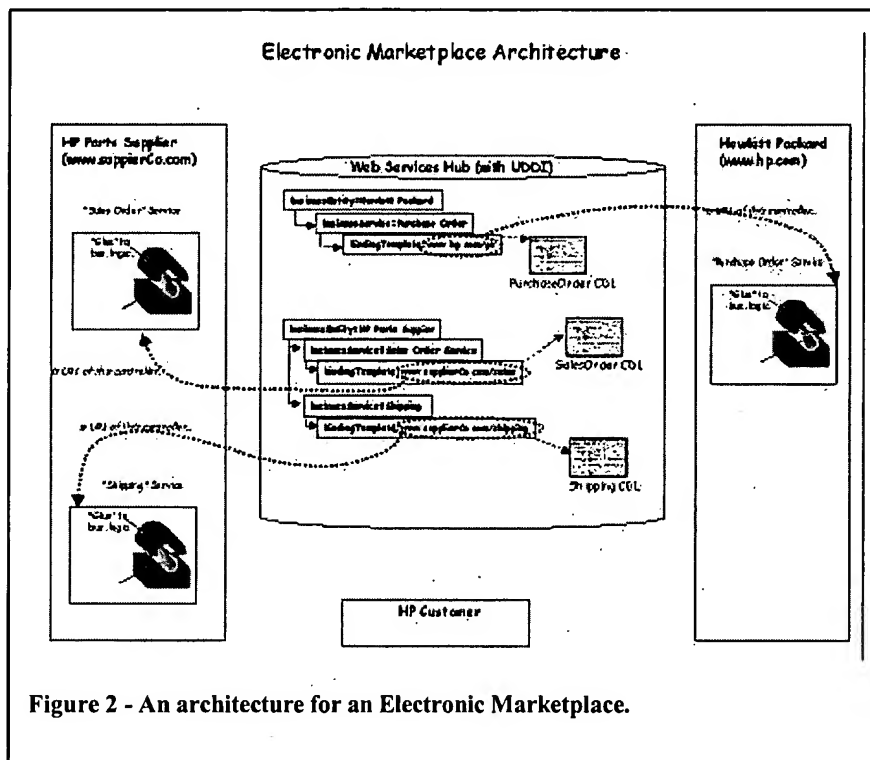
Figure 1 - Components of a mobile online printing solution.

The architecture of an electronic marketplace is sketched below in Figure 2. Note that the two architectures are quite similar -- the MNO in the mobile solution serves as the aggregating and intermediating entity, whereas the hub in the electronic marketplace serves as the aggregation point and may intermediate the access to services hosted on it. We suggest that W3C standardize the technical conventions that enable web services to inter-operate with other web services. The conventions that need to be standardized fall into two categories: infrastructural and domain specific. Infrastructural conventions should be separated into many layers that provide the functionality that essentially allow web service creation, deployment, interaction, and execution. In addition, specific web service infrastructures may also standardize common functionality that is applicable in specific domains. Two domains of particular interest in the web services space are electronic marketplaces, and mobile services. In addition, it is important for W3C to keep all the web services related conventions coordinated in order to enable a coherent platform.

3. Infrastructure for Web Services

The basic infrastructure for web services serves the same purpose that CORBA and COM serve for traditional distributed computing infrastructures. However, there is an important distinction between the infrastructure for web services as opposed to traditional infrastructures. The inter-operability problem

among web services is more challenging than the inter-operability between traditional distributed enterprises. One reason for this is that web services may be separated by firewalls and the semantics of data being communicated between them may not be uniform at the communicating ends. Another important distinction between web services and traditional distributed component models is that the binding between carious web services is looser and can occur later than the binding between various components of a traditional distributed application.



Assumptions

Note that one of the key differences between the traditional distributed computing models and web services infrastructure is that the message formats need standardization. This is in addition to standardizing the APIs needed to access the functionality. Standardizing just the APIs as the Java™ distributed computing platform has done leads to a situation where there are many incompatible implementations. For instance, at the time of writing this position paper getting two JMS implementations to communicate was still an open issue. We believe that the message formats should all be characterized as XML messages and W3C guide their definition so that different web service infrastructure implementations can inter-operate. In addition, as mentioned before, web services are fundamentally different from traditional distributed models for the following reasons:

Web services are loosely coupled: Changes to a web service should not require re-installation of software components by the users of the web service.

Web services require dynamic binding: Typically, application designers bind software components to one another at development time. Web services, on the other hand, are likely to be implemented and provided by different service providers. In addition, we must enable easy changes to the services we are

using, easy discovery of new services, of new capabilities of existing services, and of new binding or location information of services.

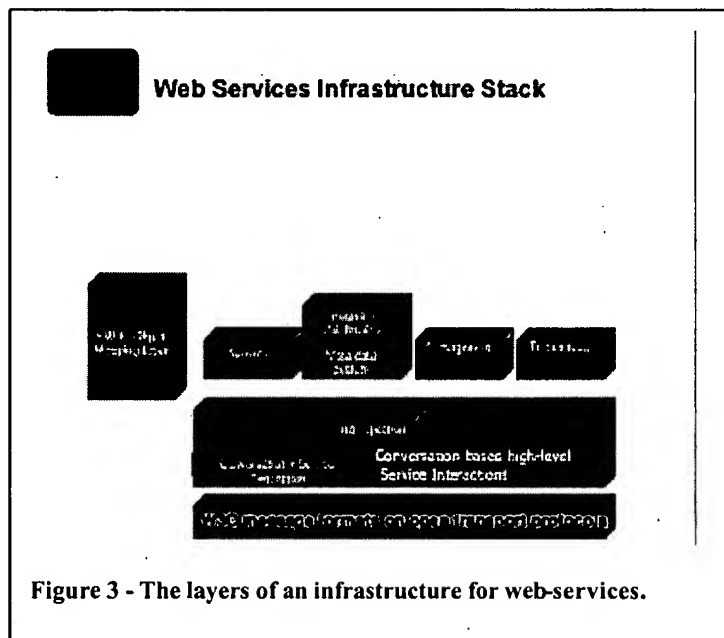
Web services communication is based on a Document Exchange Model: Traditional component frameworks support a network-object model of interaction in which objects of strictly defined types are transferred between components using a request-response interaction pattern. Cross-organizational business interactions do not fit this framework well for two reasons. The interfaces of services may need to be changed in ways that cannot be captured by simple extensions. This precludes the use of object inheritance to support the inter-operability in presence of change. Secondly, interactions can be long lived. Therefore, asynchronous exchange of XML documents is better suited for cross-organizational business transactions.

Web services in different enterprises are likely to use different semantics. The interpretation of the data communicated among enterprises is different for each enterprise. For instance, the address field of a purchase order may have different significance for the parties. If a uniform object model is used, the semantics of data often tends to be similar or homogeneous contributing to tighter coupling.

Web services in different enterprises require a distributed model of security. Security responsibilities are split amongst the enterprises. Each enterprise manages its end of the security infrastructure independently.

Web services from different enterprises may be built using heterogeneous technology stacks. Each enterprise decides on the computing infrastructure independently taking into consideration many factors.

Web service interactions must be able to traverse corporate firewalls. Traditional distributed systems are tuned for applications that are deployed within the enterprise; Web services may be deployed from behind Firewalls. They may need to access other web services across enterprises. For example, in the J2EE™ architecture, the application are written to receive input from outside the enterprise, however, the applications do not often initiate outbound requests to applications in other enterprises. In the realm of web services, accessing applications in other enterprises is the norm rather than the exception.



As shown in Figure 3, the infrastructure for web services is made up of the following layers:

The communication layer: The communication layer should be transport independent though bindings for common transport protocols such as HTTP, SMTP, etc., should be defined. Essentially, this layer answers the question: How do web services communicate with each other over standard protocols? The messaging layer provides the requisite properties of the messaging such as reliability, security, etc. In order for web services to communicate with each other they have to agree on the technology, standards and protocols for communication. They also need to agree on the syntax and semantics of data they are going to exchange. However, the data that they exchange can be classified into infrastructure parts that the infrastructure uses in order to route the message and application specific parts that the infrastructure does not inspect.

The Service definition layer. Essentially, this layer is concerned with defining how services expose their interfaces as well as the service invocation model. It addresses question such as do the web services expose a RPC like invocation model or do they expose a completely asynchronous document exchange model. The invocation model rests on the messaging model. Since the programming model for web services is more likely to be document exchange based, it is difficult to adopt an object interface definition language for the purpose of defining interfaces of web services. Another aspect to keep in mind when defining interfaces in terms of message exchanges is that the order of message exchanges plays a role in the interface definition. In addition, the security infrastructure has to provide mechanisms that allow service providers to easily determine whether the invoker has the authorization for invoking the service. It should be noted that the service definition layer provides the platform on which the other infrastructure services and functionality is built on. We at HP have invented CDL (Conversation Definition Language) as a means of expressing the interface of web services in terms of the documents that are exchanged with the service.

Introspection: The web service infrastructure has to define mechanisms that allow clients of web services to introspect web services in order to determine how they should interact with them. There are two conversations that allow services to get information about another service they are interested in. These two conversations support the dynamic interaction by allowing a functionality that is similar to the notion of introspection in traditional object systems. The ServicePropertyIntrospection conversation provides general information about the service, the conversation it supports, its provider, and how to access the service. The ServiceConversationIntrospection conversation returns the complete description of the conversations as CDL documents.

The mechanism for binding to services: This allows clients of services to determine the service they want to use in a declarative manner separate from the interface definition of the service. This essentially involves defining the meta-data of services and registry infrastructure for storing and querying the meta-data of services. Matchmaking is the process of putting service providers and service consumers in contact with each other. The matchmaker is where services that want to be dynamically discovered register themselves or their service offerings, and where services that want to find other services send their request for matches. Some of the services advertising themselves through the matchmaker will be simple end-providers, while others may be brokers, auction houses and marketplaces which offer a locale for negotiating with and selecting among many potential providers. The matchmaker is a very simple, foundational, service on which the rest of the service framework rests, and should be as neutral as possible. It is the web-spider search engine of the web services world. The matchmaker service depends on the meta-data definition system that is provided by the infrastructure. This meta-data system should be flexible and capable of defining the meta-data of a wide variety of services. Some of the requirements of the meta-data definition system are the following. It should be capable of defining the metadata for a wide variety of services. This means that if different vertical industries want to define their metadata for the services in their industry in different ways, the mechanism should allow that. It should allow the metadata to evolve in a gradual manner without changes to the backends or the enterprises that are using old versions of the metadata definition. This allows specific advertisers to differentiate their descriptions

of the goods/services they sell with characteristics that enhance their advantage over their competitors. It should be compatible with existing metadata definition mechanisms so that existing web services can be advertised at matchmakers and discovered. It should identify portions of the metadata that are searchable

The security infrastructure: This enables web services to secure various aspects of the service. Traditional PKI infrastructures may not be suited for the web services space. In addition, the security infrastructure will have to provide not only secure transport, but also higher-level access control mechanisms. Web Services or E-Services pose a unique challenge to the design of a distributed security system. At the high level the security system performs the access control, accountability, system integrity and confidentiality.

Unfortunately security systems today cannot do these things well in a distributed environment that spans multiple, independently managed security domains. Some of the specific problems that must be solved are:

- Elimination of man in the middle attacks at boundaries of security domains. Frequently security must be compromised to convert from one security model to another at a border of one or more security perimeters.
- End to End authorization and confidentiality
- Accountability
- The system must not rely upon any central security domain. There are known problems with global name spaces, interoperability of dissimilar security domain infrastructures, the need for trusted third parties at boundaries, etc. The system's objective is to establish an authorization, accountability, and confidentiality agreement between a client and service without depending upon any centralized security domain.
- The system needs to be suitable (scalable?) for a wide range of service value. A service can range from a few cents to potentially millions (billions?) of dollars. While multiple security systems could be invented, tailored to the needs of low or high value services, the challenge is to develop a single security infrastructure that can be readily adapted to either extreme.

Transaction Support: The web services infrastructure should provide support for some notion of transactions so that web services can compose other web services in a transactional manner. Traditional two phase commit protocols may not be appropriate in the context of web services. In addition, transactionality in the web services world may be restricted to providing support for atomicity only. This is in contrast to the consistency, isolation, and durability properties guaranteed by traditional distributed transaction systems. Some of the questions that need to be answered by the support for transactions are:

- Do we allow resource locking?
- What is the model for achieving transactions?
- What protocols do we need to support in order to achieve atomicity?
- Do we make compensation time-bounded?
- Does atomicity apply to conversations or isolated invocations?
- Should we enable composition of different atomicity models?

Management Support: The web services infrastructure should specify how web services are to be managed so that various aspects of the interaction among web services can be managed. There are two interpretations of manageability for services. *Manageability from a management system's perspective* refers to whether a service provides sufficient information (events, measurements, and state) and control points (lifecycle control, configuration control, etc) so that a management system can effectively monitor and control the behavior of the service. There is a second notion of manageability – *manageability from a peer service's perspective*. When two services interact with each other, one of them initiates a request (client) and the other services the request (service). From a client's perspective, a service is manageable

if the latter provides sufficient visibility and control over itself and over the transactions or conversations it executes. For example, a service that provides information about the progress of a client's ongoing transactions and an ability to escalate the speed of transactions in progress (perhaps by paying additional price) is more manageable than a service that does not. Similarly, a service that can be queried about how much quality of service (QoS) it can guarantee and that can provide response time measurements about its transactions is more manageable than a service that does not provide these features. From a service's perspective, a client is manageable if it can provide enough information about service usage back to the service. For instance, a client that can be queried about its perception or quality of experience with the service is more manageable than a client that cannot be. Similarly, a client that provides end-to-end response time measurements of its transactions back to the service is more manageable than a client that does not.

4. Domain Specific Infrastructure

We now turn our attention to some of the domain specific functionality that the web services infrastructure should standardize. There are two main domains that we believe web services have a major impact. These are: electronic marketplaces, and mobile web services.

5. Electronic Marketplaces

In the specific domain of integrating business applications across enterprises, web services can play a role in standardizing some of the interactions between businesses. The web services infrastructure provides facilities to find web services and interact with web services. However, in the business to business scenario, the negotiation and contract formation functionality has special relevance.

Negotiation: Standardizing the protocols required for various forms of negotiation such as auctioning, two-party negotiation, etc., allows web services to cleanly attach their own specific strategies while still being able to negotiate with a wide variety of parties in order to complete business transactions. The negotiation framework aims to provide infrastructure that allows two or more independent entities to interact with each other over time to reach agreement on the parameters of a contract. It is aimed primarily, though not exclusively, as a means to reach trade agreements. It can be used both by automated entities, and by users via appropriate software tools. Its value to negotiation participants is that it is a prerequisite to provide decision support or automation of the negotiation, and hence make the process more efficient. Furthermore, they can be confident that the basic rules of interaction in any negotiation are standardised, hence reducing the effort to automate many different kinds of business interactions. They are able to negotiate simple contracts, where only price is undetermined, and more complex contracts where many complex parameters depend on each other. Furthermore, the protocols provide the participants with trust guarantees, that no party has access to extra information or is able to forge false information. Its value to negotiation hosts such as auction houses and market makers is that it provides a standard framework that all potential customers can use to interact with them. However, it does not require a specific market mechanism, so allows the host to decide on an appropriate one. It not only provides standard off-the-shelf market mechanisms such as the English auction, but also allows custom mechanisms to be implemented for particular special needs such as the FCC auction for auctioning bandwidth.

Contract Formation and Business Composition: The central idea of the conceptual model for contracts is that the business relationship that motivates interactions that follow is captured explicitly in an electronic contract. An electronic contract is a document formed between the parties that enter into economic interactions. In addition to describing the promises that can be viewed as rights and obligations by each party, the e-contract will describe their supposed respective behavior in enough detail, so that the contract monitoring, arbitration and therefore enforcement is possible. The terms and conditions

appearing in the e-contract can be negotiated among the contracting parties prior to service execution. In this way businesses with no pre-existing relationships can bridge the trust gap, be able to strike deals and take them to completion. Business composition is the ability for one business to compose e-services, possibly offered by different providers, to provide value-added services to its customers. Composition of e-services has many similarities with business process (workflow) automation. A web service virtualizes the customer interaction aspects of the business processes implementing a service. In order to specify how services should be composed, we must define the interaction process associated with a service as well as the flow of service and inter-service invocations.

Conversation Definitions: defining interfaces of web services

Kannan Govindarajan, Alan Karp, Harumi Kuno, Dorothea Beringer, Arindam Banerji

Hewlett Packard Company

10450 Ridgeview Court MS 49EL-FR

Cupertino, CA 95014

Abstract

In this position paper, we advocate a novel methodology for defining interfaces of web services. This novel methodology is motivated by the realization that web-services have unique characteristics not addressed by traditional methods of defining interfaces. In particular, our methodology addresses the fact that web-services are characterized by loose coupling amongst the participating entities, as well as a message oriented interaction model. We identify various characteristics of a desirable solution.

1 Problem Statement

Web services, or e-services are applications that interact over the open internet through the use of standard protocols. In order for web services in one enterprise to interact with web services in other enterprises, we must establish technical conventions for standardizing interactions with web services. These technical conventions range from the messaging formats to interaction definitions, to properties of the interactions such as security, transactionality, etc.

Traditional distributed object models use the concept of interfaces to model interactions. This is a useful technique because the language in which an interface is defined can be completely independent of any language used to implement that interface. However, most traditional distributed object infrastructures were designed for distributed systems whose deployments were limited to within a single enterprise. They are thus suited to deploying services across organizational boundaries due to the following inherent characteristics of the inter-enterprise web services:

Web services are loosely coupled: Changes to a web service should not require re-installation of software components by the users of the web service.

Web services require dynamic binding: Typically, application designers bind software components to one another at development time. Web services, on the other hand, are likely to be implemented and provided by different service providers. In addition, we must enable easy changes to the services we are using, easy discovery of new services, of new capabilities of existing services, and of new binding or location information of services.

Web services communication is based on a Document Exchange Model:

Traditional component frameworks support a network-object model of interaction in which objects of strictly defined types are transferred between components using a request-response interaction pattern. Cross-organizational business interactions do not fit this framework well for two reasons. The interfaces of services may need to be changed in ways that cannot be captured by simple extensions. This precludes the use of object inheritance to support the inter-operability in presence of change. Secondly, interactions can be long lived. Therefore, asynchronous exchange of XML documents is better suited for cross-organizational business transactions.

Web services in different enterprises are likely to use different semantics.

The interpretation of the data communicated among enterprises is different for each enterprise. For instance, the address field of a purchase order may have different significance for the parties. If a uniform object model is used, the semantics of data often tends to be similar or homogeneous contributing to tighter coupling.

Web services in different enterprises require a distributed model of security. Security responsibilities are split amongst the enterprises. Each enterprise manages its end of the security infrastructure independently.

Web services from different enterprises may be built using heterogeneous technology stacks. Each enterprise decides on the computing infrastructure independently taking into consideration many factors.

Web service interactions must be able to traverse corporate firewalls.

Traditional distributed systems are tuned for applications that are deployed within the enterprise; Web services may be deployed from behind Firewalls.

6. Characteristics of Solution

It is our position that in the web-services world, there needs to be a clear distinction between the public interface supported by a service and its private process that implements the public interface. This distinction enables flexible and dynamic inter-operability between Web services; for example, a service

implementor can re-implement a service as long as it still supports the public interface. We base the definition of these public interfaces on the document exchange model, meaning that service interaction points are defined in terms of the documents that are exchanged with the service rather than method signature definitions.

In addition to defining service interaction points, an enterprise deploying a web service also needs to specify the valid sequences of message exchanges (interactions) that the service supports. For instance, RosettaNet PIPs define the interaction sequences that are specific to supply-chain like interactions amongst enterprises. However, RosettaNet does not support loosely coupled web services, and thus does not provide a generic mechanism for defining new interactions among services running in different enterprises. We identify that a generic, open ability to define the valid interaction sequences as part of the interface definition is necessary for enabling loose coupling amongst web-services. Such definitions would enable clients to determine dynamically the relative order in which the documents are to be exchanged to perform any unit of work with the service.

7. Outline of CDL

The Conversation Definition Language (CDL) is an XML schema for defining valid sequences of documents exchanged between web services. Conversations have been studied in the agent community, and provide a means of defining the interfaces of services in terms of the interactions that the service supports. The interaction definitions are at a level that is higher than the transport layer and allow high-level modeling of the interfaces that the service supports. For example, the details of the transport binding are the responsibility of a separate layer in our architecture and can use existing technologies such as WSDL [1] for this purpose. Essentially, CDL allows web services to model their public process in a lightweight manner. CDL also provides an extensional view of a web service in terms of the messages that are exchanged with it.

Conversation based interface definition enables loose coupling among web services by defining an asynchronous document exchange model, as opposed to a remote procedure call model, for interaction amongst web services. Web services can inter-operate with each other as long as they conform to the conversation definitions exposed by them irrespective of the implementation stack that supports the conversation definitions. CDL as described in this paper suffices to model interactions amongst two web services. We plan to extend it to support secure, mediated, and multi-party conversations.

The notion of interactions amongst services can be expressed in many ways.

One way is to express the concept of the shared interaction that abstracts away the entities in the interactions. CDL, however, specifies the view of the interaction from the viewpoint of the entities that are interacting. This means that a programmer who is programming a service can define the CDL for his service just as programmers today define the interface for the code that they write. In many vertical industries, there may be standard ways of defining conversations that can be found at an internet-wide registry. In this case, the programmer can select the conversation that she wants her service to support. This allows the clients of the service to interact with the service in a well-defined, though loosely-coupled, manner.

The design of CDL separates the mapping of external interactions between services from mappings to internal legacy business logic, which most service implementations will typically encapsulate. This latter mapping is described through business logic mapping layer documents that are outside the scope of this paper. Furthermore, the conversations that a service supports can become one of the pieces of information that is required at service registration, just as tModels for services are required when services are registered at UDDI registries today. Furthermore, the CDL definitions themselves can be listed in registries so that clients can discover the CDL definitions required to interact with services that support the CDL.

One can extend CDL in a variety of ways in order to capture various kinds of semantics. Some of the possible extensions include:

- Disconnected conversations: support for long-lived conversations with mobile entities.
- Multi-party conversations: support for conversations amongst multiple entities.
- Transactional conversations: support for atomicity and other desirable properties for interactions and conversations.
- Secure conversations: support for security.
- Mediated conversations: support for a third mediating entity that can monitor and control the execution of the conversation, like the hub-based messaging in RosettaNet specification.
- Mobile conversations: support for mobile clients in addition to the notion of disconnection.
- Support for events: the ability to define what the conversation definition will look like when support for events is added.

8. Relationship to existing standards

The conversation definition language defines the notion of interfaces and protocols of web services. Here we use the word "protocols" to denote the ordering of invocations. It presents an abstraction that is above the format of the XML message on the wire that other standards such as SOAP define.

For example, an interaction amongst the services could be a SOAP message, where the body of the SOAP message contains the document that represents the request from the client to the service. CDL defines conversations that take place on top of a messaging layer. However, CDL is independent of any particular messaging protocol.

The current version of WSDL (1.0) is an XML-based format that describes the interfaces and protocol bindings of web service functional endpoints. WSDL also defines the payload that is exchanged using a specific messaging protocol; SOAP is one such possible messaging protocol. However, neither UDDI nor WSDL currently addresses the problem of how a service can specify the sequences of legal message exchanges (interactions) that it supports. Like WSDL, CDL does not expose the service implementation. Both CDL and WSDL provide a language to define interactions amongst the entities. Unlike WSDL, CDL relies on a separate specification to define method dispatch, and relegates the binding to various message protocols to a lower layer in the stack.

Note that conversation definitions are not workflow definitions. The key difference between a conversation and a workflow is that a conversation models the externally visible interaction model of the web service. A workflow is one way to actually implement the web service.

9. References

[1] Web Services Definition Language (WSDL) available at <http://msdn.microsoft.com/xml/general/wsdl.asp>

Advertising and Discovering Business Services

Alan H. Karp, Kevin Smathers
Hewlett-Packard Laboratories
Palo Alto, California
Alan_Karp@hp.com
Kevin_Smathers@hp.com

Abstract

One of the key needs for businesses to operate effectively over the Internet is the ability to discover providers of services that they need. It is our position that the mechanisms used by existing marketplaces and *ad hoc* consortia do not fully meet this requirement because they lack the flexibility needed in the dynamic environment of the Internet. In particular, web-based services need to be able to describe themselves in new ways without undue delay. We believe that vocabularies are a representation of ontologies that are well suited to the business needs. A framework that allows for easy creation, dissemination, and evolution of vocabularies, while accommodating industry standard vocabularies better meets this need.

1. Business Need

As business services move to the web, it becomes increasingly important to automate the way buyers and sellers find each other. Advertising at the Super Bowl is not going to attract software. Clearly, some sort of automated advertising and discovery mechanism is needed. However, blind searching on keywords results in too many false misses as well as too many false hits. What is needed is a context that provides semantic meaning to the search terms.

An ontology provides semantic content to an attribute, allowing parties to agree on the meaning of the value associated with that attribute. For example, in an electrical engineering ontology, GATES might be an integer denoting the number of electronic gates per square inch. In a landscape architecture ontology, GATES might be a string denoting the style of gate. There are many ways to represent ontologies. Examples include hierarchical taxonomies and vocabulary expressed as name-value pairs. We feel that the latter best meets the needs of e-businesses.

Business interactions on the web use ontologies in a way that differs from many common uses, leading to a set of requirements on the infrastructure.

- Adaptability is key. However the ontologies are defined, it must be possible to describe new kinds of things without undue delay. Waiting for the next version of a standard ontology constitutes undue delay.

- Ontologies must be widely available. It doesn't do a business any good to describe itself in an ontology unknown to its potential customers.
- Translation from one ontology to another related ontology is critical. Simply translating between languages is enough to justify this requirement, but the ability to cross market segments that have their own way of describing the same thing enforces it.
- Fixed values registered in a catalog service may not be adequate. For example, advertised prices may depend on who is doing the search or when the search is done. Hence, *dynamic* attributes are needed.
- Complex descriptions and queries must be supported. A business is more than just its list of products and their prices. It is a combination of those factors plus its business practices and procedures.
- Delayed discovery, in which the searcher is notified when a suitable service is found whether or not the initial search succeeded, is needed in many scenarios.

2. Other Ontology Efforts

We have been discovering things on other computers as long as there have been networks, probably even before. The Internet and the growth of commerce on it have dramatically increased both the need for discovery and the difficulty in providing it. While a survey of all ontology efforts would be of value, it is beyond the scope of this position paper. However, understanding why these efforts fail to meet the needs of business being conducted on the Internet is important for understanding our position. We've picked some representative examples to illustrate the problem.

Probably the first efforts attempted to index the entire web. The Alta Vista search engine provides a full text search of every web page it has indexed. A problem with this approach is that you get no hits or 40,000, a phenomenon that has been facetiously called the *Alta Vista effect*. Yahoo also indexed the web, but with human indexers. However, searching requires following a fixed hierarchy, something better for humans than software. Other hierarchical indexing schemes, such as CoS Naming and LDAP, provide wildcarding that avoids some of these problems, but they don't provide any special mechanisms to support ontologies. You can always make a particular node in the hierarchy an ontology node in which points below it are defined in terms of that ontology. Unfortunately, the very feature that makes it easier for software to search, wildcarding, makes it impossible to enforce the use of the ontology.

Object systems, such as CORBA or Jini, provide a means to find objects that implement specific interfaces. However, a business service is more than the interfaces it supports. While Jini is built on top of JavaSpaces, a very general search engine that

could be used to provide more complex description and discovery, there is no specific support for ontologies. VerticalNet supports a large number of trading communities. Each has a specialized ontology to enable businesses within that community to find each other. However, there is only one ontology per community, so that additions must wait for approval from a central authority.

More recent efforts to provide description and discovery frameworks are UDDI and ebXML. UDDI Version 1 allows businesses to describe themselves in one or more standard taxonomies, such as NAICS and UNSPSC. However, these taxonomies are not flexible enough for all situations. For example, it is difficult to decide if the category "computer service" is for hardware or software. Version 2 of UDDI, currently being designed, allows new taxonomies to be introduced. However, allowing the description of a business service in detail is not one of UDDI's goals. It is only intended to provide a first level filter; further discrimination is done in direct communication with the service provider. Another standard in the making, ebXML, is similar to UDDI Version 2 in that it allows categorization in different taxonomies. However, ebXML includes more of the other aspects of doing business on the web than does UDDI.

E-speak was designed for doing business in the dynamic environment of the Internet. Business services in an e-speak environment are constructed by specifying the job that needs to be done rather than how the job is to be done. Thus, a business process that needs a billing service describes the properties it is looking for rather than naming a specific billing service. Hence, a rich, flexible description and discovery mechanism was critical to making e-speak useful. It's not surprising that e-speak incorporates many of the features needed for discovery of business services on the web. Our position is strongly influenced by our experience with e-speak, both in understanding the most valuable features of e-speak vocabularies as well as in knowing what extensions would be most useful.

3. *Dynamic Vocabularies as Discoverable Services*

We believe that businesses will want to use industry standard ontologies, but they need a way to meet special needs on a time scale shorter than these standard ontologies can be updated. We propose a framework for defining *vocabularies* to meet this need. A vocabulary is a particular representation of an ontology that allows a business service to be described as attribute-value pairs. Further, a vocabulary can be advertised as a business service itself in another vocabulary. A very simple *base vocabulary* understood by all is needed to ground the recursion.

This mechanism is best illustrated by an example. Say that I am interested in finding a billing service to incorporate into my business processes. I find general business services vocabularies by doing a lookup in the base vocabulary. I use those vocabularies to find the ones related to business processes. I can then look for the exact

service I'm interested in. Note that I may turn up more vocabularies, which I can then use to extend my search. More complex cases involving detailed business services might well go through more levels. However, at the end of the process, I will have found a vocabulary that is rich enough to describe the services in sufficient detail. Of course, the vocabulary creator decides how much detail is sufficient. If a business finds that it needs an extension, all it needs to do is create a new vocabulary and advertise it in the previous vocabulary.

The vocabulary framework must support certain features that fall into 4 categories.

Specification

- It should be possible to specify both the vocabulary and the query in XML. Using XML makes it easier for independently coded applications to use the vocabulary. It also means that a specification for doing translation is in place.
- The vocabulary mechanism must allow for user-defined attribute types. These types can be defined in terms of other user-defined types, but at some point the definition must end in a rich set of architected types. One such type should be a service description in terms of another vocabulary.
- Mandatory attributes, those that must be included in every service description, allow the vocabulary creator to enforce consistency standards.
- Multi-valued attributes are a must. It must also be possible to define attribute values as a range. These requirements can be combined by specifying the allowed attribute values as a constraint on the type.

Advertising and searching

- Businesses must be able to advertise in multiple vocabularies.
- The search language must support complex queries, including inequality and substring matching
- The search mechanism must provide support to express arbitrary boolean relations between terms from different vocabularies.

Evolution of vocabularies

- The vocabulary framework must permit the translation of one vocabulary to a related one.

- It should be possible to evolve a vocabulary without invalidating previous uses.
- Allowing inheritance from another vocabulary will make introducing new vocabularies easier.
- Often two or more equivalent vocabularies may be developed. It is important to distinguish between two references that refer to such equivalent vocabularies as opposed to independent references to the same vocabulary.

Creation and control

- The owner of a vocabulary must be able who can modify its definition.
- Controlling who can advertise in a vocabulary is one way that marketplaces can provide a degree of trust. Controlling who can search using a vocabulary provides the service provider with a degree of trust in the client.
- The vocabulary should carry information about the creator of the vocabulary in order for the user to have sufficient trust in the matching rules it contains.
- The creator of the vocabulary must be able to define the matching rules, and these matching rules should be definable per attribute, not just per value type.

This last point needs some explanation. Consider a string-valued attribute. What constitutes a match on a “less than” query? Is it collating sequence? In what language? Is it substring? Case sensitive? Starting at the beginning of the string? The creator of the vocabulary has the semantic knowledge to understand the meaning of the attributes and answer these questions, and the answers may be different for different attributes.

Once such a general vocabulary mechanism is in place, it can be used for other purposes. Events can be defined in terms of a vocabulary. Publishers and subscribers can find each other by advertising in the event’s vocabulary. Event state can be specified as attribute values in the vocabulary and subscription filters can specified as constraint expressions in the vocabulary. A vocabulary can also form the basis of online negotiation and contract formation. Each multi-valued attribute can be treated as a clause in a contract, and negotiation can proceed to settle on values in the contract.

4. References

This section provides links to the pages of the technologies referenced in this document.

1. Alta Vista: <http://altavista.com>
2. CORBA: <http://www.corba.org>
3. ebXML: <http://ebxml.org>
4. E-speak: <http://e-speak.net>; <http://e-speak.hp.com>
5. Jini: <http://www.sun.com/jini>
6. LDAP: <http://www.openldap.org>
7. UDDI: <http://uddi.org>
8. VerticalNet: <http://www.verticalnet.com>
9. Yahoo!: <http://www.yahoo.com>

Requirements for Automated Negotiation

Claudio Bartolini¹, Chris Preist¹, Harumi Kuno²

Hewlett-Packard Labs

⁽¹⁾ Filton Road, Stoke Gifford, Bristol, BS34 8QZ, UK

⁽²⁾ 1501 Page Mill Road, Palo Alto, CA 94304, USA

Email: claudio_bartolini@hp.com, chris_preist@hp.com, harumi_kuno@hp.com

1. Introduction

The increasing importance of business to business electronic trading has driven interest in automated negotiation to soaring heights. In recent times, web-service enabled electronic marketplaces have been displacing proprietary trading solutions. Looking at this trend, we foresee a need for a general software infrastructure that enables independent entities to interact using multiple forms of negotiation. This infrastructure would cover a variety of aspects, including defining a general protocol for negotiation (including the definition of the actors, roles and phases of negotiation); defining a taxonomy and a language for negotiation rules to cast the general protocol into one that embodies the desired market mechanism; defining a language to express negotiation proposals.

Negotiation has been for decades a central subject of study in disciplines such as economy, game theory, and management. When discussing negotiation, it is important to distinguish between *negotiation protocol* and *negotiation strategy*. The protocol determines the flow of messages between the negotiating parties, dictating who can say what, when and acts as the rules by which the negotiating parties must abide by if they are to interact. The protocol is necessarily public and open. The strategy, on the other hand, is the way in which a given party acts within those rules in an effort to get the best outcome of the negotiation for example, when and what to concede, and when to hold firm. The strategy of each participant is therefore necessarily private. In this document we concentrate on the requirements for architecting software enabling automated negotiation, therefore we discuss protocols and not strategy.

Existing approaches to architecting software enabling automated negotiation provide either ad-hoc software for particular market mechanisms or proprietary solutions. We take an open approach by defining a standard protocol for interaction among the participants in the negotiation process. Our protocol is defined independently from the negotiation rules embodying the particular market mechanism that the negotiation host wants to impose. Instances of negotiation rules will be used to cast the general protocol

into a specific one that embodies the desired market mechanism. This approach is general with respect to a wide variety of market mechanisms, from one-to-one negotiation to auctions and double auctions.

2. Value proposition

Our position is that we need to design a standardized infrastructure that allows two or more independent entities to interact with each other over time to reach agreement on the parameters of a contract. This infrastructure is aimed primarily, though not exclusively, as a means to reach trade agreements. It can be used both by automated entities and by users via appropriate software tools.

The value of such a framework to *negotiation participants* is threefold. First, the framework frees participants from having to develop their own negotiation infrastructure, providing prerequisite services such as the provision of decision support and support for the automation of the negotiation process. Second, the infrastructure enforces the standardization of basic interaction rules, allowing participants to be confident that basic rules of interaction in any negotiation will be followed. For example, participants will be able to negotiate simple contracts, where only price is undetermined, as well as more complex contracts involving multiple complex and interdependent parameters. Third, the protocols provide the participants with trust guarantees, ensuring that no party has access to extra information or is able to forge false information.

The value to *negotiation hosts*, such as auction houses and market makers, is that by providing a standard framework that is independent of any specific market mechanism, they will increase the number of potential customers who can interact with them. The infrastructure would allow the hosts to select an appropriate market mechanism. It would provide standard off-the-shelf market mechanisms (e.g. English auction), and also allow custom mechanisms to be implemented for particular special needs (e.g. the FCC auction).

3. Requirements

We identify the following requirements for a negotiation protocol that would meet our goals:

- Be sufficiently formal that automated entities can interact using it.
- Support negotiation about simple and complex objects.
- Be sufficiently general that a variety of different market mechanisms (e.g. 1-1 negotiation, combinatorial auctions, exchanges) can be expressed as specific instances of it.

- Support security mechanisms and protocols that enable participants to do business in a trusted way.
- Allow, but not require, the existence of a third party to arbitrate a given negotiation (e.g. an auctioneer in an auction.)
- Support existing ways of doing business, as well as permitting more radical approaches in the future.

4. What needs to be defined

4.1. A general protocol for negotiation that can support a wide variety of market mechanisms

A negotiation protocol provides a means of standardizing the communication between participants in the negotiation process by defining how the actors can interact with each other. We therefore base the definition of our protocol on conversation orchestration protocols such as CDL [1] or WSDL [2].

We propose that the protocol should be general enough to support a wide variety of market mechanisms. Therefore the protocol should be based on the common aspects of the various market mechanisms that it wants to support. That is, define the negotiation process as an exchange of negotiation proposals followed by a phase of agreement formation. The two phases will often be intertwined for some market mechanisms.

Designing the protocol requires the definition of:

1. The roles played by the actors involved in negotiation processes
2. The phases of the negotiation process (e.g. admission, proposal submission, agreement formation)

4.2. A taxonomy of rules of negotiation

As noted above, the protocol is defined independently from the negotiation rules embodying the particular market mechanism supported by the negotiation host. The rules for negotiation will then cast the general protocol into one that can be used to embody a particular market mechanism.

Examples of types of rules for negotiation would be rules for deciding on the well-formedness of a negotiation proposal. Another example is rules regulating the alternation of participants in submitting proposals. Again, there will be rules dictating the visibility aspect of proposals in many-to-many negotiation, i.e. who among the participants is entitled to see a submitted negotiation proposal, and so on. Rules will be needed for supporting mechanisms such as zero knowledge proofs to

avoid revealing private information in the course of negotiation.

4.3. A language to define rules of negotiation

The language to define the rules of negotiation should be standardized. The idea is to have a declarative language for expressing rules in a way that participants to negotiation can reason about them. The declarative layer would then be mapped to reusable software components implementing the logic expressed by the rules. These components would be plugged in the orchestration infrastructure for the protocol to be cast to embody a desired market mechanism.

4.4. A language to express negotiation proposals

The format to express negotiation proposals has to be standardized. The requirements to be satisfied by a candidate language would be:

- Support for ontology and namespaces
- High degree of expressiveness
- Ability of expressing less than fully bound specifications
- Ability of expressing constraints over ranges of possible values as well as definite values of a specification
- Loosely supporting types and some degree of inheritance
- Support for complex queries
- Support for complex matching

RDF [3] and DAML-OIL [4] are promising candidate languages

5. References

- [1] Govindarajan K., et al. *Conversation Definitions: A way of defining interfaces for web services* – submitted to W3C Workshop on Web services, 11-12 April 2001, San Jose, CA
- [2] <http://msdn.microsoft.com/xml/general/wsdl.asp>
- [3] <http://www.w3.org/RDF>
- [4] <http://www.daml.org/2000/12/daml+oil-index.html>

Towards the Electronic Contract

Michal Morciniec, Claudio Bartolini, Brian Monahan, Mathias Sallé

Hewlett-Packard Labs

Filton Road

Stoke Gifford

Bristol BS34 8QZ

UK

Email: michal.morciniec@hp.com, claudio.bartolini@hp.com,
brian.monanhan@hp.com, mathias.salle@hp.com

1. Introduction

In recent years we have witnessed an explosion of business applications exploiting Internet as a communication medium. Initially, on-line catalogues and shop fronts have been deployed followed by auction sites and finally business-to-business (B2B) infrastructures [Sculley 1999]. Electronic marketplaces are e-commerce infrastructures that aggregate potentially large numbers of buyers and sellers and allow them to interact according to a variety of market mechanisms such as request for quotes, reverse auction or exchange resulting often in significant cost saving. They rapidly evolve towards trading of web services rather than commodity goods.

As each enterprise tries to maximize its goals, conflicts of interests are certain to appear. Possible concerns [Favier 2000] range from security of transactions, fairness of the market mechanism, anonymity, identity of business partners to service performance. We feel confident that adequate solutions can be provided for most of the technology related concerns, however, a technology gap exists in addressing concerns that have their roots in the societal aspects of business interactions.

In the real-world the contracting process together with the established institutions and adequate enforcement and trust mechanisms provide an answer to these concerns. We assert that reification of this process will be required in order to enable enterprises establish and manage dynamically changing business relationships that can be formed in the electronic marketplaces.

2. Electronic contract

A contract is a statement of intent that regulates behaviour among organizations and individuals. An electronic contract is its reification in software that can be instantiated as a set of obligations that are fulfilled between parties, refused or waived as future events occur. Because the contract parties are assumed to be autonomous and self-interested, conflicts will occur, and an appropriate resolution mechanism is required.

An electronic contract can be viewed through transformations that are applied to it during its lifecycle in the electronic marketplace. Further, there is an information viewpoint that shows (document) objects that can be observed at the beginning and termination of each stage. Conceptually, the lifecycle can be split into the three stages of contract drafting, formation, and execution.

Contract drafting phase – Given the contract template model, the drafter role constructs an instance of the template. In this phase the contractual roles, abstract business interactions and contractual situations are specified. Furthermore, if the drafter acts as a regulator, rules and constraints can be added which should be adhered to during the contract execution phase. The template typically has a number of free variables that are agreed upon in the next phase.

Contract formation phase - Participants assume contract roles and negotiate the details of their responsibilities. The negotiable variables of the contract (deadlines, order of actions) become fixed, and concrete business interactions are bound to the abstract ones defined in the template. The relationships between contract parties are created and are captured in the contract statements using the policy expressions that imply obligations and rights of parties.

Contract execution phase –Actual delivery of contract consideration happens. Typically this phase constitutes service or goods delivery, invoicing, bill calculation, presentment and payment. The interactions between the parties may be monitored for their conformance to the terms of the contract.

3. Scope of the proposed work

Standardization effort is required to enable enterprises to interact with each other and electronic marketplaces. The scope of the effort is broadly related to the provision of a contracting framework in the context of electronic marketplaces. More specifically it includes the following:

1. Definition of the structural model for electronic contract (means to capture contractual commitments) that includes relevant information objects related to the contract during its lifecycle

2. Definition of the main phases and roles involved in the contract lifecycle and transformations applied to it. In particular,
 - Contract formation (contract drafter, contract negotiator);
 - Contract fulfillment (service provider, service consumer);
3. Specification of the protocols used by the relevant roles to achieve these transformations. In particular,
 - Protocol used for contract formation;
 - Protocol used for contract fulfillment;
4. Interfacing of the contracting framework components with other components in the enterprise.

4. References

[Sculley 1999] Sculley A.B., William W., Woods A., 1999, "B2B Exchanges : The Killer Application in the Business-to-Business Internet Revolution", ISI Publication.

[Favier 2000] Favier J., 2000, "eMarketplaces Face the Law", the Forrester Report, October 2000, <http://www.forrester.com>

Security Requirements for Web-Services

Hewlett Packard Position Paper to the Worldwide Web Consortium Workshop on Web Services, April 11th and 12th.

Author: Mike Jerbic, Hewlett Packard Co. email: Mike_Jerbic@hp.com

Abstract:

As XML document exchange models become the de facto standard for users and programs to interoperate with web services across multiple security boundaries, end to end security becomes either meaningless or at best extremely difficult. Today's XML security "solutions" center around document encryption and signature: that is securing the confidentiality and integrity of the XML document itself. Missing from the solution suite is an interoperable standard to authorize access to services and a framework for accountability should one or more components of a service invocation fail to perform. The W3C should initiate an activity to create standards for defining, discovering, and exchanging authorization and accountability information within XML document conversations.

Problem Statement

HP has taken a position that XML document oriented conversations are appropriate and necessary to exchange information needed to discover, exchange necessary information, invoke, and receive electronic services over the Web. For XML conversations to become practical for the exchange of personal information, service invocation authorization, and service participant accountability, security must be designed into the conversation framework from the beginning.

Security services in use today do not satisfactorily address the needs of an open, web-oriented document exchange model for electronic service delivery. They are deficient today in these ways:

- Security services are single domain oriented. E-services by design cross multiple domains. Examples include service composition across enterprises and the overall Business to Business electronic commerce problem. Even though PKIs (such as Verisign) appear to span security domains, they by virtue of being a "trusted third party," simply establish their own domain. Service providers and consumers must be able to establish a security context independent of so called trusted third parties.
- Security systems require unnecessary information, the exposure of which is

a global privacy, public safety, and security issue. Many times names, contact information, and other non-essential personal information is required to support legacy identity oriented security services. Adversaries misuse this information routinely. A global e-services infrastructure must allow for communication of privacy policy, the sharing of sensitive information, and recourse in the event privacy agreements are not kept. Some specific industries that are facing critical privacy problems yet also have the business requirement to exchange private information over the web in the conduct of their e-services are:

- United States Health Care delivery (HIPPA regulation requirements).
- United States multinational enterprises who must comply with the European Union Privacy Directive for the protection and use of European citizen information
- Banking and Financial Services transactions
- B2B and B2C service invocation and payment
- And many others...

Privacy and Security expectations must be part of the agreement [XML conversation] between service providers and service consumers. They cannot be ambiguous. XML conversations should have the flexibility of being private conversations.

- Security systems do not scale well and may actually prevent service providers from entering markets. Security systems often do not scale in
- Cost. Cost can be measured in several ways including legacy system integration or porting costs and the cost of ownership of central security servers or services.
- Time. Security servers that map names to authorizations, for example, may require several hours up to days to vet new entries or to make permission or role changes granting authorizations to new services. Open, web based dynamic service providers require much faster, easier to change authorization systems. Real time manageability of security databases is a significant impediment to the dynamic advertising, discovery, and invocation of e-services.

Solution Requirements

Overall HP's position is that XML documents must be able to include security context

information that is discoverable, invocable, and interoperable. The solution needs to include:

- *Authorization.* Computers do not need identities to perform requested services. Computers need authorizations. Authorizations should be embeddable or attachable to XML service requests. Authorizations range from simple payment (credit card number, digital cash...) to a certified, tamper resistant authorization message originating from an authorization authority the service recognizes.
- *Accountability.* While computers only need authorizations, people need methods of determining accountability. Usually this is achieved through an access control list, which maps authorizations to an identity and an audit trail of activity. Other forms of accountability information are possible and should be explored.
- *Manageability.* Multiple methods of authorization management must be supported. Three primary ones are:

Mapping names (identities) to authorizations. This is one of the most common forms of managing Service access to named individuals. Also known as an Access Control List, mapping names to authorizations works well in small communities where identities are commonly known and used. This method does not scale well for large communities where name collisions can occur. An example is below:

Authorization Access Control list example for three identities.

Name	Service 1	Service 2	Service 3	Service 4 ...	Service n
Mike	Yes	Yes	No	Yes	No
Terry	No	No	Yes	No	No
Eliot	Yes	Yes	Yes	Yes	Yes

Mapping names to roles, mapping roles to authorizations. An extension of the access control list is the creation of roles and privileges. Individual identities are mapped to roles, which in turn are mapped to specific authorizations. This scheme is somewhat more manageable for services that have a large client base (large number of names) or a distributed client base, where individual authorization is delegated. An example is below.

Authorization Access Control list example for three roles.

Role	Service	Service 2 Add User	Service 3 Bill User	Service 4 Credit	Service n
Subscriber	Yes	No	No	No	No
Admin	No	Yes	No	No	No
Back Office	No	No	Yes	Yes	No

Identity to Role Map

Identity	Subscriber	Admin	Back Office
Ted	Yes	No	No
Mike	No	Yes	No
Terry	No	No	Yes

Direct Authorization from the service provider. There are times when a determining authorization does not require checking a map of identity to authorization, as in the two examples described above. A service can simply grant access to an entity. This approach is best documented in the Simple Public Key Infrastructure (SPKI). In this system a service requestor is given an authorization to use the service by the service. When a service is invoked, the requestor presents a tamper resistant message authorizing the access to the service provider. No external authorization database check is required, since the authorization verifier has all the information required in the service invocation request. Direct authorizations may be delegable.

- *End to End Security.* Service invocations and responses should be able to be confidential (encrypted) between the requestor and the service provider, without relying upon any intermediary third party such as a web server, portal, etc. Applications do exist where intermediaries may need to examine the traffic flow between service provider and consumer, and the security architecture should be able to support this.

- *Support of existing security mechanisms, protection of IT investment already in place.* Security contexts need to have the flexibility of using legacy security infrastructure components such as X.509 identity certificates, S2ML, SSL, and integration with web browser security.
- *Discoverability.* XML conversations need to support the exchange of security context information, and the requirements to access a service must be discoverable by the service requestor. A conversation infrastructure must be developed that includes security.
- *Scalability.* The solution must support the service issuing its own authorization to a client without the need of an additional trusted party (such as a CA). Certificate Authority or Authorization Authority oriented solutions such as S2ML, SSL should be supportable as well for those service providers who want/need to leverage them.
- *Privacy.* A method for interoperable exchange of privacy intentions is strongly desired. (Leverage / extend P3P?)
- *Tamper Resistance.* The security architecture must provide for tamper resistance (most likely through digital signatures, XML DSIG?).
- *Non-Repudiation.* Tamper resistance, which is technically feasible, must not be confused with non-repudiation, which is not. Digital signature technology can present some evidence of the signer's intent, but non repudiation and the legal trustworthiness of exchanged documents must be governed not only by available technology, but also by law and contractual agreements.

Relationship to Other Standards

To address the requirements above, many already proven security standards must be considered and supported in the proposed new standard where they add value. While not intended to be an exhaustive list of standards to consider, the following serves as a starting point. It is important to note that the W3C already has experience in developing and recommending security solutions, and several of the standards have origins in W3C activities.

SSL secures a web browser to a web server. In the case of a portal of many services, confidentiality offered by SSL ends, with the client relying on the web server to end service security outside of the client's control. SSL is not end to end in a distributed service world.

S2ML provides an authorization authority model that must be queried every time an authorization is required. S2ML is a standard in process at the OASIS standards organization.

XML Encryption provides confidentiality of an XML document, but it does not provide

authorization information in a standard, interoperable way.

X.509 PKI – OK if the service provider can use name to authorization maps.

SPKI – OK if the service provider can't use name to authorization maps.

Session Layer Security (SLS) – While not strictly a standard, HP has invested uniquely in a distributed authorization system to secure end to end Java applications. The security mechanism of HP's E-Speak, SLS could be extended to XML. HP awaits the opportunity to work with the W3C to propose and extend this technology.

P3P – A description of a site's privacy policy that can be downloaded and automatically compared to the user's preferences.

XML DSIG – Provides digital signatures to XML documents. This functionality should be considered for inclusion as part of the XML security standard to support tamper resistance.

A Framework for Business Composition

Andy Seaborne, Eric Stammers, Fabio Casati, Giacomo Piccinelli, Ming-Chien Shan

Today, the Internet is not only being used to provide information and perform e-commerce transactions, but also as the platform through which services are delivered to businesses and customers. More and more companies are rushing to provide all sorts of services on the Web, ranging from more "traditional" on-line travel reservations and directory services to real-time traffic reports and even outsourcing of entire business functions of an organization, such as IT or human resources departments.

Business composition is the ability for one business to compose e-services (possibly offered by different companies) to provide value-added services to its customers. Composition of e-services has many similarities with business process (workflow) automation. An e-service virtualises the customer interaction aspects of the business processes implementing a service. In order to specify how services should be composed, we must define the interaction process associated with a service as well as the flow of service and inter-service invocations. In this paper we refer to a business process that composes e-services as *e-process*.

We first outline the trend to business composition and e-processes, and then set out some requirements on standards to be developed to support e-processes.

E-Processes and E-Services

E-processes are typically designed, developed, and deployed by enterprises that want to compose internal capabilities with third-party capabilities, either for internal use or to expose them as (complex, value-added) e-services to customers. Note that, while the e-services involved in an e-process may belong to several companies, the e-process is company-specific. Its definition is typically known to and controlled by a single company. For both e-processes and e-services, we do not envision the need for companies to expose internal details of how they run their business. Still, companies should be able to express in a standard format the interaction aspects for their service offer as well as for their service needs. A common language is crucial for the assessment of the compatibility between the interaction processes offered by an e-service provider and those expected by the designer of an e-process.

The effectiveness and efficiency of business processes impact directly the profitability of a company. It is in the best interest of e-service providers as well as e-service consumers to understand the operational requirements for their cooperation. An e-process defines the interactions between the company owning the process and the e-service providers involved in its implementation. In particular, an e-process defines the orchestration activity needed to enable the cooperation between the e-service providers. As traditional processes were designed around the operational model of customised

business applications, e-processes should be designed around e-services. A clear understanding of the business interaction model of an e-service is paramount.

The Evolution of e-Processes

Phase 1 : integration of existing internal assets

Today, enterprises are automating their processes to reduce costs and improve process execution quality and speed. Process automation and management technologies enable the separation between business logic, resource logic, and application logic. Process can be controlled, managed, and evolved separately from the applications. Still, in this phase, resources are internal to the enterprise.

Phase 2 : static integration with partner processes on a case-by-case basis

By incorporating e-services provided by business partners into an e-process, an enterprise can create processes that utilize external resources. However, in this phase, service selection and invocation is still performed in an ad-hoc way, and require preliminary agreements (from a business, legal, and technical perspective) between the cooperating companies.

Phase 3 : dynamic integration with negotiation, with companies

Beyond such static use of external services, fully dynamic e-processes make decisions each time they are executed in order to invoke the best available service that can fulfill the customer's needs. The tradition design-deploy cycle of phases 1 and 2 has changed to a per-instance set of decisions.

In the remainder of the paper we focus on dynamic e-processes, since these are the ones that can provide the best value and are in line with the philosophy of the e-service environment. In particular, which aspects of e-services should be standardized in order for users to define e-processes that compose e-services and execute them in a fully dynamic and efficient way.

Enabling Standards

We now focus on dynamic e-processes, since these are the ones that can provide the best value and are in line with the philosophy of the e-service environment. In particular, which aspects of e-services should be standardized in order for users to define e-processes that compose e-services and execute them in a fully dynamic and efficient way. Some of these standards are under development by industry consortia or standardization bodies, while others still need to be developed.

Service Metadata

In order to stay competitive, service providers should offer the best available service in every given moment to every specific customer. Similarly, e-processes should be able to compose the best services available when the process is executed. Hence, when defining an e-process, we need to be able to describe the type of service we want, and let the system discover which specific e-service can satisfy the e-process needs. In order to be able to select e-services and understand their characteristics, a standard for service description is needed. This standard should allow the definition of the e-service properties, so that the e-process can determine its suitability for satisfying the business needs.

We also note that we do not envision the definition of a common, global ontology that can fit every e-service description. In fact, while businesses will certainly want to use standard ontologies when available, they will often need to define attributes faster than allowed by standardization bodies. Hence, the standardization effort here should allow the definition as well as the discovery of ontologies.

Service Interface

The description of an e-service should include the specification of the e-service interface. In order to support e-processes, a standard way of defining this interface is required, so that the e-process execution engine can determine which operations are available on the selected e-services and how to invoke them. It is also desirable to have a description of the semantics of each operation, to understand, for instance, whether the invocation of an operation commits the invoker to a payment or to send a registration form.

Note that services are typically complex entities, offering several methods (operations) to be invoked as part of their interface. For instance, an e-music service may allow users to browse or search the catalog, to listen to songs, or to buy songs. The service provider may want to impose constraints on the order in which these operations are invoked. Hence, a simple, IDL-like definition of the interface may not be sufficient to specify how the interaction with a dynamically discovered service should be conducted. We refer to the set of interactions (i.e., invocation of operations) with an e-service as *conversation*. In order to be able to correctly interact with a selected service, a standard description of the supported conversations is also required.

Transactional Processes and Services

A desirable feature in workflows as well as in service composition is the ability to execute parts of the process in an atomic fashion, i.e., so that all of it is executed or the effect of partial executions can be (semantically) "undone". A typical example is the reservation of a trip, where the customer would like to "atomically" book the flight and

the hotel. If, for instance, the flight could be booked but no hotel room is available, then the flight reservation needs to be "cancelled" or "undone". A standard description of the transactional properties of an e-services would give the e-process engine the ability to perform such atomic executions and to identify how to interact with services when performing "commits" and "rollbacks".

Negotiation

Advanced service composition systems may provide facilities for negotiating with the discovered e-services before accessing them, and for reverting to other services if the negotiation fails. A standard for negotiation is required so that service composition tools may have embedded negotiation capabilities. In this way, the benefits of negotiations could be made easily available to e-process designers and users.

Security

Many services may require security credentials before allowing interactions. A standard need to be defined so that e-process definition languages allow the definition of the appropriate security certificates to be used, and e-process engines know how to authenticate and get authorizations for service executions.

Contract

A contract forms the basis of an e-process between businesses. The contract identifies the roles and responsibilities of the parties, the obligations and deliverables. With dynamic integration, it become important to explicitly represent services. In earlier phases, when there was an explicit integration phase, the issues in a contract would have been dealt with. With contract choices being made during the execution of an e-process, the agreements entered into will need to be recorded.

Service Discovery

The previous subsections have discussed mechanisms to enable interaction with an e-service once it has been discovered. However, mechanisms to discover e-services are required, so that the e-process can dynamically look for e-services during each execution. Hence, standards for identifying and querying service directories in order to retrieve information on e-services and their properties are needed.

Non-Requirement

In the previous section we have stated some requirements for dynamic e-processes. In this section we want to state a non-requirement because we believe it to be unnecessary.

There is no need to provide an explicit representation of the business network dynamically generated to deliver the value of a service. In other words, the services and processes used by one service are not exposed to the service clients. The dynamic nature of e-services means that the exact choices of which other services to use may not have been made at the time a contract is entered into.

The service client neither knows nor is concerned about the internal details about how a service meets its obligations. Each service has a partial view of the business network, no single point having a global view of the entire network.

The service provider is free to search and invoke other services that best meets its needs. Companies do not expose their internal operations or be constrained to do business as prescribed by some "global" process definition. Instead, companies want to expose services, but be free to implement such services the way they want.

Transactional Conversations

Svend Frolund and Kannan Govindarajan

Hewlett-Packard Company

email: svend_frolund@hp.com, kannan_govindarajan@hp.com

1 Introduction

We believe that web services should make their transactional properties available to other web services in an explicit and standard way. Transactional properties should be part of a service's interface rather than a hidden aspect of its backend. The transactional behavior of a service can then be exploited by other services to simplify their error-handling logic and to make entire business-to-business interactions transactional. However, such business-to-business transactions are challenging to implement because they span multiple companies and because the underlying transaction protocols execute over wide-area networks.

It is fundamental for web services to communicate through conversations. A conversation is a potentially long-running sequence of interactions (document exchanges) between multiple web services. For example, a manufacturer may engage in a conversation with a supplier and a conversation with a shipper to carry out the activity of purchasing parts. In many situations, the backend logic triggered as part of these conversations may be transactional. For example, it may be possible to arrange for parts to be shipped, and then later cancel the shipment (if the parts have not actually been sent yet). Cancelling the shipment is an example of a compensating transaction, it compensates for the initial transaction that arranged the shipment. Since the notion of conversation is fundamental to web services, the exportation of transactional properties should fit with conversations, giving rise to transactional conversations.

In the Internet setting, atomicity is the most important aspect of transactions. Atomicity means that the effect of a transaction either happens entirely or not at all. We also refer to this as all-or-nothing semantics. If a service A knows that a conversation with another service B is atomic, then A can cancel the conversation and know that B will cleanly revert back to a consistent state. Furthermore, A can rely on the B's transactional behavior to ensure state consistency in the presence of failures, such as logical error conditions (e.g., shipping is impossible) or system-level failures (e.g., the crash of a process or machine).

Services should expose their transactional behavior in a manner that facilitates composition of transactions from different services. For example, it should be possible for the manufacturer to compose a transactional conversation with the supplier and a transactional conversation with a shipper into a transactional activity that includes both conversations. The advantage of creating these multi-conversation transactions is that the manufacturer gets all-or-nothing semantics for the entire end-to-end purchasing activity: if shipping is not available, the order placement is cancelled. This is a very powerful notion, that we believe will significantly reduce the complexity of

programming business-to-business activities between multiple web services. Composite transactions provide a notion of “clean abort” for entire business-to-business activities. Moreover, having a standard notion of transaction allows us to build generic software components that perform the transaction composition.

2 Atomicity

We discuss different ways for transactions to be atomic. As terminology, we introduce the notion of a transaction outcome, which is either commit or abort. The outcome is abort if the effect of the transaction is “nothing.” The outcome is commit if the effect is “all.”

2.1 Two-Phase Commit and Compensation

If we execute two atomic transactions, their combined effect is not necessarily atomic: one transaction may abort and the other may commit, which means that the combined effect is neither all nor nothing. If we create a composite transaction from two constituent transactions, we need to ensure that either both constituent transactions commit or that both constituent transactions abort. There are two traditional ways to ensure this. One way, called two-phase commit, is based on the idea that no constituent transaction is allowed to commit unless they are all able to commit. Another way, called compensation, is based on the idea that a constituent transaction is always allowed to commit, but its effect can be cancelled after it has committed.

With two-phase commit, transactions provide a control interface that allows a transaction coordinator to ensure atomicity. One incarnation of the control interface is the XA specification [3]. Essentially, the control interface provides a prepare method, an abort method, and a commit method. The coordinator calls the prepare method to determine if a transaction is able to commit. If the transaction answers “yes,” then the transaction must be able to commit even if failures occur. That is, the transaction is not allowed to later change its mind. If all transactions answer “yes,” the coordinator calls their commit method, otherwise the coordinator calls their abort method.

With compensation, there is no additional control interface. Instead each “real” transaction has an associated compensating transaction. With compensation, a coordinator can ensure atomicity for a number of constituent transactions by executing the transactions, and if any transaction aborts, the coordinator executes the compensating transaction for all the transactions that have committed.

2.2 Discussion

Although both two-phase commit and compensation can provide atomicity for composite transactions there are trade-offs between the two methods. Compensation is optimistic in the sense that the coordinator only enters the picture if something bad (e.g, abort) happens. With two-phase commit, the coordinator enters the picture even if all transactions commit. The coordinator always calls prepare and either commit or abort for any transaction. On the other hand, two-phase commit always provides a point after

which a service can forget about a transaction. Once the transaction is instructed to commit, the service can forget about the transaction. With compensation the service has to be able to compensate forever. The ability to compensate may or may not require the service to maintain persistent state. Of course, there are hybrid models where compensation is bounded by time or the occurrence of events (such as receiving a notification).

In practice, few systems use two-phase commit in the Internet context. One reason is that, with two-phase commit, a service exposes transaction control to other services. If a service answers “yes” in response to a prepare request, the service has to be able to commit the transaction until instructed otherwise by the coordinator (which may be another service). Few services are willing to export such transaction control in a loosely-coupled system. Another reason for the limited use of two-phase commit is that composite transactions may be long running. If we want transactions to span entire business-to-business activities, we have to accept the possibility that transactions may run for a long time. With two-phase commit, a constituent transaction cannot commit until the composite transaction can commit. Thus, a fast service may be forced to wait for a slow service.

If we want to support two-phase commit, we need a protocol that allows flexible designation of the coordinator role. For example, a given web service may be willing to play the role of participant in certain situations, but may insist on playing the role of coordinator in other situations. If we have a conversation definition language, such as CDL [1], we can capture this distinction through different conversations. A service can export a coordinator version and a participant version of the same logical conversation.

We believe compensation is a fundamental notion of atomicity for web service, and in the remainder of this paper, we shall only consider compensation. This does not reflect a position against two-phase commit, but is merely to simplify the discussion.

3 Isolation, Durability, and Consistency

Traditional database transactions satisfy the ACID properties (atomicity, consistency, isolation, and durability) [2]. Unlike traditional database transactions, we do not believe that transactions, in the context of conversations, should necessarily provide isolation. Isolation is concerned with controlling the concurrent execution of transactions to provide the illusion that concurrent transactions execute one after the other in some (indeterminate) serial order. Isolation is unnecessarily strict for Internet transactions. This is evident from many Internet sites that provide transactions without isolation. For example, sites, such as Amazon.com, provide transactional semantics in the form of compensation (cancelling an order within a given time limit) and do not provide isolation. Besides being unnecessarily strict in many cases, isolation is also costly because transactions may be long running, and providing isolation for long-running transactions hampers the overall performance.

To continue the comparison with database transactions, we would expect the “primitive,” constituent transactions to provide durability and consistency. Durability means that transactional updates are made persistent if the transaction completes successfully. Consistency means that a transaction takes the system from one consistent state to another. The durability and consistency of constituent transactions follows from the transactional properties of the backend logic in web services. We do not believe that “composite,” multi-conversation transactions should provide any global notions of durability or consistency beyond what the constituent primitive transactions provide. In other words, we do not rely on any particular notion of durability or consistency when we compose primitive transactions into composite transactions.

4 Describing Transactional Properties

We outline briefly what it may take to describe, and thus export, transactional properties of web services. The starting point for our discussion is the assumption that services communicate through explicit conversations. If a service exports a description of its conversations—the conversations it is willing to engage in—the question is how the service can specify the transactional properties of those conversations. The specification makes explicit to other services how the service is transactional. The specification should communicate the following aspects of transactions:

- **Demarcation.** We need to describe which parts of a conversation are transactional. If we consider a conversation as a sequence of interactions, we need to identify the transactional sub-sequences of those interactions. At one extreme, the entire conversation may be transactional. But other possibilities may exist as well. For example, only a single interaction may be transactional, or an identified sub-sequence may be transactional. In general, a single conversation may have multiple transactional parts.
- **Outcome.** To exploit the transactional behavior of a service, we need to know the outcome (commit or abort) of its transactions. One way to communicate the outcome of a transactional conversation to other services is to associate a particular outcome with a particular point in the conversation. For example, a specific interaction may denote abort and another interaction may denote commit. If the conversation reaches an interaction that indicates abort, the parties of the conversation know that the outcome is abort. We need to describe which interactions indicate abort and which interactions indicate commit. Notice that we can also use document types instead of interactions to indicate the outcome of transactions.
- **Compensation.** We need to describe how transactions can be cancelled or compensated for. For example, a conversation may have a particular document that triggers compensation, or different documents may trigger compensation at different points in the conversation. To initiate compensation at a given point in a conversation, sending a compensation document must be a legal interaction at that point in the conversation, and we must be able to generate the appropriate

compensation document. Notice that compensation may not be possible at any point during a transactional conversation, so we need to know both how and when compensation is possible.

If a service exports a description of its conversations in the form of an XML document, we can think of the description of the transactional properties as a companion document.

5 Requirements

To conclude, we outline basic requirements for web service transactions.

We want our notion of web service transaction to fit with conversations. Conversations provide the context for transactions: transactions take place within conversations, and we talk about transactional conversations. The integration of conversations and transactions have consequences for the transaction model. Because conversations can be long-running, so can transactions. The transaction protocols, such as two-phase commit and compensation, involve communication between web services. These communications should be first-class members of the conversations between web services. For example, if we have a conversation definition language to describe conversations, we should use that language to describe the transaction protocols as well.

We want to support compensation as part of the transaction model. With two-phase commit, transactional web services rely on an external entity, a transaction coordinator, to communicate the transaction outcome to them. Such reliance on external entities may not always be appropriate in loosely-coupled systems. Compensation does not introduce the same level of reliance on external entities. Our position is not against two-phase commit, but rather in favor of compensation: two-phase commit protocols may be appropriate in certain situations. If we have a transaction model that supports both two-phase commit and compensation, we have to address the issue of “mixed-mode” transactions—transactions whose constituent transactions are based partly on two-phase commit and partly on compensation.

In general, regardless of the choice of transaction model, we want to support a decentralized, peer-to-peer model for transactions. For example, we do not want to assume the existence of a centralized transaction coordinator. We do not want to prevent a centralized notion of coordinator, we simply do not want to rely on one. Notice that the notion of a transaction coordinator may be relevant for both two-phase commit and compensation. A central coordinator might make sense in conjunction with compensation. This coordinator would then gather the outcomes of the various constituent transactions and execute compensation transactions as necessary.

We need to address the issue of trust between the web services that participate in a transaction. Both two-phase commit and compensation assumes that the various parties

are well-behaved (or trusted). For example, two-phase commit assumes that participants vote “honestly” and that they do as instructed (commit or abort). Furthermore, the notion of compensation also assumes that a participant actually executes a compensating action if instructed to do so. With two-phase commit, each participant also trusts the coordinator to be in control of the protocol—the protocol is inherently asymmetric because the coordinator knows the outcome before any of the participants. Since trust is a general issue for web services, we assume that some other mechanisms are put in place to deal with trust in a general sense. In terms of transactions, we need to integrate with those general mechanisms to handle trust. It is unlikely that we can treat trust as a completely orthogonal issue to transactions.

References

- [1] K. Govindarajan, A. Karp, H. Kuno, D. Beringer, and A. Banerji, “Conversation Definitions: defining interfaces of web services,” submitted to the 2001 W3C workshop on web services.
- [2] J. Gray and A. Reuter, “Transaction Processing: concepts and techniques,” Morgan Kaufman Publishers, 1993.
- [3] Distributed Transaction Processing: The XA Specification, X/Open Snapshot, 1991.

A Peer-to-Peer Service Interface For Manageability

Vijay Machiraju, Akhil Sahai
E-Service Management Project
E-Services Software Research Department
HP Laboratories, 1501 Page Mill Road, Palo-Alto, CA 94034
{vijaym, asahai}@hpl.hp.com

1. Introduction

Various interfaces and protocols are being defined for seamless composition and inter-operation of web services. Some of these are used by web services to discover each other (e.g., UDDI [1]), some of these are used by services to express their functionality to each other (e.g., WSDL [2]), and some of these are used to invoke operations on each other (e.g., SOAP [3]). While all of these are essential for easy composition, there are still many aspects that services need to agree upon in order to interoperate. One prominent example of such an aspect is manageability.

There are two interpretations of manageability for services. *Manageability from a management system's perspective* refers to whether a service provides sufficient information (events, measurements, and state) and control points (lifecycle control, configuration control, etc) to a management system so that it can be effectively monitored and controlled. There have been many efforts – Common Information Model from Distributed Management Task Force [4], Manageability Service Broker from Open Group [5], and Java Management Extensions from SUN [6] to name a few – for standardizing the interface and protocol between managed services and management systems. So far, management systems and these standards have been focused on managing enterprise applications. As web services become more prevalent, cross-enterprise management of federated services will become increasingly important.

There is a second notion of manageability – *manageability from a peer service's perspective*. Figure 1 shows two peer-to-peer services interacting with each other. An

interaction is any form of communication between two services. This could mean a single-step transaction (e.g., a login to a book-selling service), a sequence of related transactions (e.g., logging in, adding books to shopping cart, and checking out), or even business-level processes perceived by the client that may involve manual steps (e.g., the process from ordering books to final delivery of the books). For every step in an interaction, one of the two services initiates the request and the other executes the request. We refer to the service that initiates the request as the *consumer service* and the one that executes the request as the *provider service*. The role of a service could change over the course of an interaction.

From a consumer's perspective, a *provider service is manageable* if the latter provides sufficient visibility and control over itself and over the interactions it executes. For example, a provider that provides information about the progress of a consumer's ongoing interactions or an ability to escalate their speed is more manageable than a provider that does not. From a provider's perspective, a *consumer service is manageable* if it can provide enough information about its service usage back to the provider. For instance, a consumer that can be queried about its perception or quality of experience is more manageable than a consumer that cannot be. Manageability interfaces capture the functionality that should be offered by providers and consumers to each other in order to be manageable.

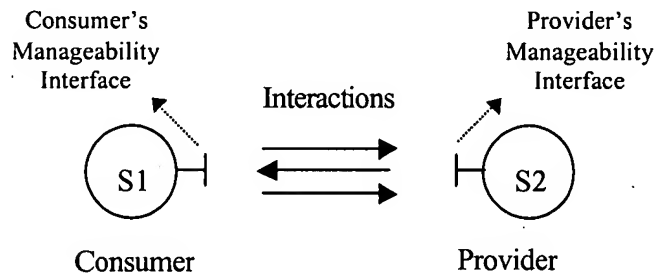


Figure 1: Peer-to-peer services, their roles, and manageability interfaces

There have been no efforts to standardize interfaces and protocols of this nature between web services. In this paper, we first motivate the need for such interfaces by explaining their benefits. We then list a set of elements that should be part of these interfaces. Finally, we conclude with some suggestions for how to approach the standardization.

2. Need for Peer-to-Peer Service Manageability

We envision a world where services will be capable of advertising, discovering, composing, and using each other to execute their functionality. Providing interfaces for manageability between services would help in realizing part of this vision.

In the discovery-phase or pre-composition phase, a consumer can use manageability interfaces of providers to obtain valuable information about their quality of service - performance, availability, reliability, and service-level guarantees. Brokers and other discovery services help clients select services. However, the selection criteria are currently limited to price and functionality. Standardizing manageability interfaces will facilitate negotiation and selection based on a whole new set of features. With this additional information, consumer services will be able to assess if they can guarantee end-to-end quality to their own clients taking into account the quality levels guaranteed by their provider services. When services are composed, it is not only the functionality that gets composed; the service-levels, performance, and availability get composed as well. Hence it is only natural for services to expose these details so that a consumer can make informed decisions before composition.

In the execution phase or post-composition phase, a consumer that uses a manageable provider has better visibility and control over the latter; and hence will be able to make better decisions. For example, a consumer that needs faster service can escalate its interactions to the next tier of service temporarily. Similarly, it can request long-lasting transactions to be shutdown so that they can be executed on a different service. Consumers will be able to compare different services for their quality and performance.

Third party rating services can be formed that just collect and sell ratings of various services for their quality. A need for third party mediators would also arise for mediating and monitoring compliance of service level agreements between other services. A real life analogous example would be the credit rating companies and the manner in which they settle disputes. However, the dynamics in case of web services would be different and so the third party mediators have to monitor the compliance and take corresponding actions in real time.

Being manageable is advantageous to the provider too. Manageability is a new differentiator for service providers. Higher levels of visibility and control can be provided to valued customers or at higher prices. We have seen various examples of adhoc manageability offered by various services to be more successful. For example, Fedex offers tracking of their shipping transactions (example of visibility). Most of the on-line e-commerce sites provide some form of a cancellation feature so that a service or product can be cancelled after being purchased. Web services have already realized the importance of accountability and guaranteeing service levels. A standardized manageability interface helps in projecting all these features of a web service.

One of the major drawbacks of current instrumentation and management technologies is the lack of an end-to-end solution. Once a service crosses a service provider (into a client or a supplier), he or she has no visibility or control on what happens with the interactions. Management systems internal to a service provider find it difficult to analyze if a failure or service-level violation is attributed to internal business logic or to a supplier. Having manageability interfaces on all services helps in solving this problem and in facilitating cross-enterprise management systems.

3. Elements of Manageability

Manageability is measured in terms of two factors – *visibility* or ability to observe and monitor, and *controllability* or ability to influence and change. One way to classify the functionality offered through a manageability interface is to differentiate elements of visibility from elements of control. Another dimension of classification is *service-level* versus *interaction-level*. Service-level interfaces are used to obtain information about or execute actions on the overall service. Interaction-level interfaces are used to obtain information about or control specific interactions between services.

As we have already explained, a service could act as a provider, or as a consumer, or both. The type of manageability depends on the role of the service. Table 1 shows examples of functionality that could be offered by a provider to be manageable. Table 2 shows some elements of consumer-side manageability interface.

Table 1: Elements of a server-side manageability interface

Service-level Visibility	Interaction-level Visibility
Nature of visibility and control supported.	Expected time an interaction will take to execute before submission.
Quality levels – performance, availability, and reliability – that can be guaranteed by the provider.	Expected time an interaction will take while it is in progress. Track an interaction in progress.
Quality levels currently being guaranteed to the consumer.	Amount of time an interaction took to execute.
History of quality levels with the consumer.	History and statistics about past interactions.
History of quality levels with all the consumers.	Quality levels being guaranteed on certain types of interactions to the consumer.
Generic service metrics such as number of current consumers, average turn-around time etc.	
Service specific metrics – e.g., books sold per second for a book-selling service.	
Service-level Control	Interaction-level Control

Change current quality levels being guaranteed to the consumer. For e.g., change the consumer from bronze to gold customer.	Abort an interaction. Suspend an interaction. Resume a suspended interaction.
Report a consumer-perceived service-level violation to the provider. Ask for an explanation. May result in compensation according to the contract.	Change the desired quality level for an interaction.

Table 2: Elements of a consumer-side manageability interface

Service-level Visibility	Service-level Control
Nature of visibility and control supported.	Slow-down the request rate.
Quality levels perceived by the consumer.	Fail-over to a different service.
Quality levels currently being guaranteed to the consumer.	
Interaction-level Visibility	Interaction-level Control
Perceived quality levels for a particular interaction or a class of interactions.	Re-issue or restart an interaction. Suspend an interaction.

4. Realizing the Vision

Realizing service manageability would require involvement and standardization on two fronts:

1. Define a set of standard terms, conversation definitions, and communication messages to support manageability of web services. Services first need to agree upon terms such as quality of service, contracts, and transactions. The next step would be to define conversations that should be supported by services to be certified manageable. For example, conversations for negotiating quality of service and for canceling transactions should be standardized. These conversations should cover all the types of visibility and control that can be offered by web services in a generic manner. Technologies such as XML, WSDL, and current management standards such as CIM would be helpful in defining these vocabularies and conversations.
2. Standardize extensions to current communication protocols such as SOAP to include manageability information. Services could communicate valuable

information about interactions to each other by exploiting the messages that they exchange. For example a message from a service can include expected time of completion as part of its response. Headers that capture information such as quality level expected and conversation context can be standardized for every request. Similarly, quality level delivered, response time information, and conversation context can be part of every response header.

Coordinated efforts are also needed in the area of security for web services. A good foundation of security is a pre-requisite for services to be able to offer visibility and control over their functionality.

5. References

- [1] Universal Description, Discovery, and Integration, <http://www.uddi.org>
- [2] Web Services Description Language, <http://www-106.ibm.com/developerworks/library/w-wsdl.html>
- [3] Simple Object Access Protocol, <http://www.w3.org/TR/SOAP/>
- [4] Common Information Model, <http://www.dmtf.org/spec/cims.html>
- [5] Manageability Services Broker, <http://www.opengroup.org/management/msb.htm>
- [6] Java Management Extensions, <http://java.sun.com/products/JavaManagement/>